Technical Research in Advanced Air Transportation Concepts & Technologies (AATT)

ATM Modeling and Simulation Architecture Study

AATT RTO 70

November 14, 2001

Prepared By

Science Applications International Corporation
Simulation and Information Technology Operation
1100 N. Glebe Road, Suite 1100
Arlington, VA 22205

Technical POC
Jesse S. Aronson, P.E.
703-907-2553 / aronsonj@saic.com

Prepared for NASA Ames Research Center under Contract NAS2-98002

ATM Modeling and Simulation Architecture Study

AATT RTO 70

Table of Contents

ATM Modeling and Simulation Architecture Study

AATT RTO 70

Executive Summary

The nation's air traffic control system faces many challenges in accommodating continuing growth in air traffic, necessitating development of new operational concepts for air traffic management. Design of such novel concepts can be greatly facilitated through the use of simulation tools to address a variety of questions including system-level policy assessments, national economic impact of new technologies, regional flow performance, infrastructure constraints and human performance. Recognizing the growing requirement for large-scale, NAS-wide simulation tools, NASA commissioned a study of simulation run-time architectures. The objective of the study was to provide background information required to assess the software architecture that will be required to create a framework for a simulation system that can support simulation of the entire NAS and all elements within it to allow exploration of system-wide impacts. The study included two major tasks. The first task was to survey several government simulation communities to identify applicable simulation architectures, while the second task involved evaluating technologies identified under Task 1 for their applicability to three Use Cases including 1) performance of conceptual trade evaluations covering many issues and metrics; 2) provision of detailed evaluations from many viewpoints of changes to the system prior to their implementation; and 3) conduct of real-time and non-real-time analyses of system-wide performance.

The study formulated a reference architecture and found that all three Use Cases could be satisfied by instantiations of that architecture. The backbone of the postulated architecture is the High Level Architecture for Modeling and Simulation (HLA) which was developed by the Department of Defense to address many of the same challenges in scalability, composition, integration and reuse now faced by NASA and which has achieved widespread acceptance within the DOD. The study also found that to varying degrees the components of the simulation architecture could be satisfied by existing technologies, perhaps with some adaptation to the particulars of the NAS domain.

The major challenges identified in implementation of the architecture lie in augmenting the current capabilities of the HLA to better support the requirements of analytical simulation and in fostering development of a collaborative, architecture-based approach within the NASA simulation community.

1 Introduction

1.1 Study Goals

The nation's air traffic control system faces many challenges in accommodating continuing growth in air traffic. Among these challenges are to accommodate growth in airspace system utilization while preserving and enhancing system safety, provide airspace system users with flexibility and efficiency in the use of airspace resources and reduce system delays. Accomplishment of these goals requires creative design and enhancement of airspace operations, and these design activities need to be supported by design tools including robust, flexible simulations.

The Aerospace Operations Modeling Office at NASA Ames is engaged in several activities in the area of novel NAS operational concepts, requiring non-real-time modeling of the Air Traffic Management (ATM) system. Current activities, under the Advanced Air Transportation Technologies (AATT) Project and the Quiet Aircraft Technology (QAT) Program, model specific tools or functions within segments of the National Airspace System (NAS). Consideration of new concepts requires adequate and credible models to satisfy three Use Cases as follows: 1) performance of conceptual trade evaluations covering many issues and metrics; 2) provision of detailed evaluations from many viewpoints of changes to the system prior to their implementation; and 3) conduct of real-time and non-real-time analyses of system-wide performance.

Simulation in support of these Use Cases will address a variety of questions that include system-level policy assessments, national economic impact of new technologies, regional flow performance, infrastructure constraints and human performance. Further, there must be feedback up and down between these five levels so that policy decisions derived at the highest level are accommodated in the more detailed levels, such as human performance.

Study of new operational concepts carries with it several key requirements. First, since many of the envisioned changes have NAS-wide impact, the tools used to evaluate these concepts must be NAS-wide in scope. To do this, the tools must support mixed fidelity, with insets of high fidelity (in, for example, regions, models, or human behaviors) where needed. Second, since new concepts by definition are a departure from current practice, the tool set must be flexible to easily support changes in models. The modeling system will need to incorporate existing codes and so should include mechanisms for legacy code integration.

The intention of this study is to provide the background information required to assess the software architecture that will be required to create a framework for a simulation and modeling system that can support simulation of the entire NAS and all elements within it to view system-wide impacts. The study includes two major tasks. The first task is a survey to identify applicable simulation technologies. This task included a starting list of technologies from several communities. The areas to be surveyed specifically included architectures used by Defense Advanced Research Project Agency (DARPA), Department of Defense (DOD), Department of Transportation (DOT), and other agencies that could be adapted for ATM applications. It shall included the DOD High Level

Architecture Run-Time Interface framework that is proposed for the FAA's Aviation Integrated Reasoning Modeling Matrix (AIRMM), the Georgia Tech Reconfigurable Flight Simulator, the SPEEDES and SWARM simulation engines and the FAA Aviation System Analysis Capability. During the course of the study the list was expanded to include several other DOT technologies, COTS simulation frameworks and FAA domain modeling efforts.

The second major task focuses on evaluation of these technologies against NAS-wide analytical simulation requirements and synthesis of one or more simulation architectures to satisfy the three domain Use Cases. For each technology, both the strengths and limitations of the candidate technologies were assessed and their place in the required architectures was identified.

This study focuses on run-time aspects of simulation. As shown in Figure 1.1.1, the use of simulation as part of an analysis effort starts with planning and setup tasks, continues through execution, and finishes with analysis and archiving of results. In addition, independent of any particular study there are long-term tasks to build and maintain a simulation system, including development of the execution environment and model and data management. Each of these phases in simulation utilization has its own characteristics and requirements. The direction for this study was to focus on simulation execution, that is, the pieces required to run the simulations, and so the other process steps in the simulation lifecycle are treated only where they intersect with run-time.

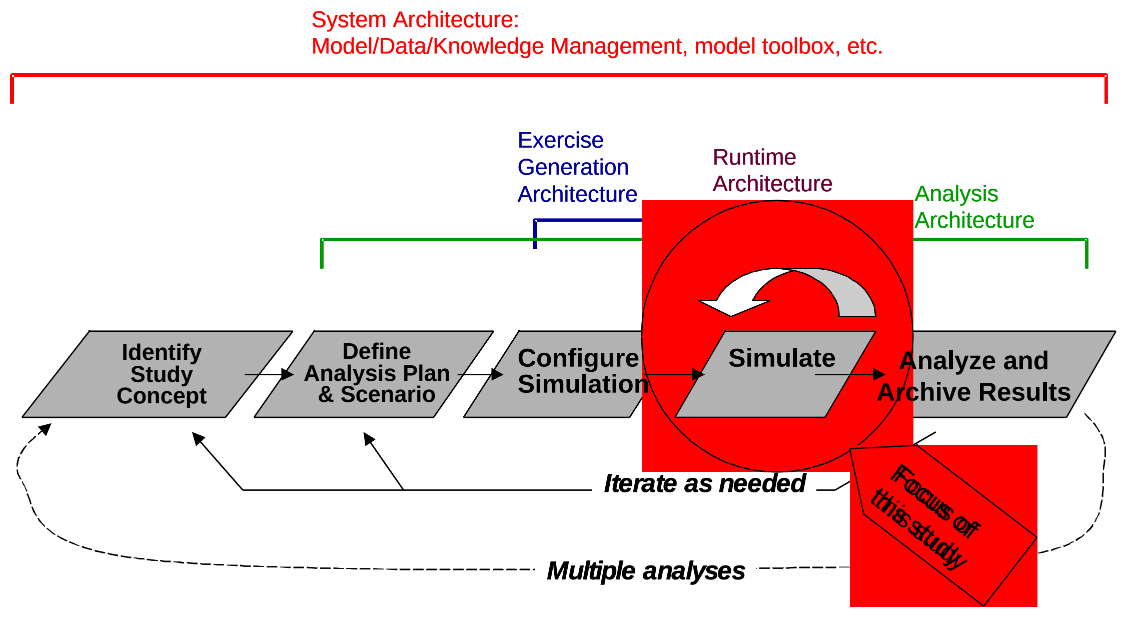


Figure 1.1.1: Simulation Lifecycle – a family of architectures

1.2 Methodology

A system engineering methodology was used in performing the study. Any assessment is contingent upon having a set of evaluation criteria and therefore as a first step the Use

Cases were expanded to determine the requirements they impose on the simulation tool kit. The results of this requirements analysis are given in Section 2.

The second step in the study was formulation of a reference simulation architecture based on the Use Case requirements. This conceptual reference architecture focuses on the top-level functions that a simulation run-time system must provide. The goal of formulating the reference architecture was to provide a consistent framework within which to evaluate the capabilities of each of the candidate technologies without delving into the details of particular technologies or physical instantiations. Section 3 describes the top-level reference architecture.

With the establishment of a reference architecture, the next step in the study was to evaluate existing technologies to determine their applicability in implementation of the various architectural components. Section 5 provides evaluations of each of the candidate technologies, including a mapping of each technology into the reference architecture. Finally, Section 5 describes a recommend architectural approach for ATM simulation based on the requirements derived in this study and the capabilities of available technologies.

2 Scenario Analysis

2.1 Use Cases

The direction for the study included three types of modeling applications, or Use Cases, corresponding with types of analyses performed by NASA: 1) performing fast-time conceptual trade evaluations covering many issues and metrics; 2) providing non-real-time detailed evaluations from many viewpoints of changes to the system prior to their implementation; and 3) conducting real-time human-in-the-loop analyses with virtual simulation/simulator components. The three Use Cases share many common attributes, however their varied characteristics result in unique requirements for each Use Case. This section examines the Use Cases to determine their individual and shared requirements.

2.2 Fast-time Conceptual Trades

The first Use Case is that of *Fast-Time Conceptual Trades*. This application typically involves using low fidelity, rapidly configurable, fast-executing models. The goals of simulation at this level are to gain an understanding of the impact of a new concept or technology, to refine partially defined concepts and to provide broad-brush evaluation in support of policy development.

An example of this Use Case would be evaluation of the national flow and safety impacts of the Automated Airspace Concept (AAC). [Er2001] In this application the example inputs and excursions might include minimum separation and the percentage of aircraft equipped for AAC. Example metrics collected could include the separation achieved, location-specific and national flow statistics, flight time and path statistics.

This Use Case has several key characteristics. First, scenarios can be up to NAS-wide in scope, that is, thousands of concurrent flights, ATM sectors, and other NAS elements. The simulation infrastructure must be able to support this large number of simulated entities, including multiple aspects of each (e.g., physical movement through both en route and terminal airspace as well as financial implications of routing changes), albeit at low fidelity. Second, despite the breadth of scenarios, this type of application requires rapid turnaround and evaluation of excursions, meaning that scenario and simulation setup must not be overly complex.

In current practice, this type of simulation typically consists of a number of simulations strung together via integrated data (for example, a capacity model feeding a delay model feeding a cost model and so on). In the future simulations may continue to be executed in this mode, however the architecture should also support concurrent execution of simulations.

2.3 Fast-time Detailed Evaluations

The Fast-time Detailed Evaluations Use Case involves applying detailed models to gain close understanding of the impact of a new concept or technology. The goal of this type of simulation is to perform detailed concept analysis within the context of the entire NAS. Consequently, this application may contain mixed fidelity models, that is, detailed representation of the concept under study surrounded by a lower fidelity context. The boundaries between fidelities can be geographic (e.g., a higher fidelity sector in the midst of a lower fidelity center), functional (e.g., high fidelity representation of TRACON airspace surrounded by lower fidelity en route airspace models) or physical (e.g., mixed fidelity representations of aircraft dynamics). Scenarios for this type of application range from localized scope (e.g., a single airport or sector) up to NAS-wide scope.

An example of a Fast-time Detailed Evaluation might be a benefits assessment of Direct To (D2) routing under current and future demand scenarios and varying weather. In this example aircraft movement through some part of the airspace would be modeled at high fidelity in order to track flight path and time statistics under different routing regimes.

This Use Case has a number of key characteristics. First, the combination of scope and fidelity required for detailed evaluation scenarios can lead to high computational requirements and therefore to the use of multiple CPUs to support the scenario. The multiple CPUs can take the form of high performance computing (supercomputers or clusters of computers) or multiple independent computers working together (but possibly running a heterogeneous mix of simulations) to form a representation of the airspace.

Second, the use of mixed fidelities implies the ability to transfer representation of simulated entities between the different simulations which compose the simulated scenario. For example, a low fidelity en route aircraft may need to transition into a higher fidelity simulation as it crosses a boundary into high fidelity TRACON airspace.

Third, Detailed Evaluations can have large input and output data volumes. Typically, Detailed Evaluations make use of complex input data, for example, historical ETMS flight data and RUC wind values, and produce detailed measures and metrics such as detailed trajectories, flight time over route, point of closest approach and conflict statistics.

2.4 Real-Time Virtual Simulations

The last Use Case is *Real-Time Virtual Simulations*. The key differentiators of such simulations are that they typically run in real-time and involve human actors who play roles in the virtual world created by the simulation. Human players typically interact with the simulation via partially or completely immersive environments, that is, ones that mimic one or more aspects of the user interface and/or sensory environment of equivalent real-world stations. Consequently, Virtual simulations often include components such as Vertical Motion Simulators, cockpit or controller workstation mockups or other large-scale immersive environments such as NASA's Future Flight Central 360° airport tower simulator.

Real-Time Virtual Simulations are used for a number of purposes, including training, human factors analysis (evaluation and design of human/machine interfaces), human performance studies (evaluation of human performance under varying conditions, stimuli and workload) and evaluation of operating procedure changes.

Virtual simulations typically focus on a more limited domain scope than the other two Use Cases based on the fact that a human participant can only directly observe a small part of the NAS at any given time. On the other hand, Virtual simulations must represent provide a richer representation than is found in the other Use Cases, including visual, aural, motion and haptic models. Thus, Virtual simulations typically contain very high fidelity models of small portions of the NAS, augmented by medium fidelity models to provide operational context (for example, an immersive cockpit simulator linked with medium fidelity simulations of nearby aircraft). Likewise, since these simulations often have the goal of observing the behavior of the human participants, data capture includes not only simulation data but also human interaction with the system (activity traces, audio and video recording of participants, etc.).

The first key characteristic of this Use Case is the ability to operate in close synchronization with real, or "wall clock" time so that the human participants cannot perceive any difference or delay relative to real-time. The second characteristic is the need to support the multi-sensory representations needed to create believable virtual environments. This has an implication on the volume of data needed for initialization and the quantities of data passed around during run-time. The third characteristic relates to data collection, where a large number of parameters may need to be collected from each simulator, possibly in coordination with audio and video streams. Mitigating the increased data requirements of these two characteristics is the fact that only a small portion of the NAS is typically represented.

2.5 Use-Case Driven requirements

Figure 2.5.1 summarizes the characteristics of the three Use Cases. It is understood that NASA is separately performing more detailed studies on the computational requirements of various types of simulations; the present table serves only as a general categorization of simulations required to support the Use Cases. The table has a number of rows representing the various characteristics of the architectures, as follows:

- *Distributed processing*: The number of computers and separate simulations involved, connected by networks.

- *Variability of Scenario Models and Data*: The extent to which scenario data and configuration typically change from run to run of the simulation. This provides an indication of how much scenario/simulation setup is required for each run. This is also an indication of how much the composition of models will change (e.g., "for analysis 'A' we need a 6 DOF motion model but for analysis 'B' we only need a 3 DOF model").
- *Complexity of input data*: The volume and number of attributes for both databases (terrain, weather, ETMS data) and scenario-specific data.
- *Volume of data collected*: The volume of data collected from each run of the simulation.
- *Volume of data exchanged*: The volume of data that would be exchanged between different inter-connected simulations at run-time.
- *Total computational complexity*: A measure of the overall computing power needed. Differs from volume of data in that computational complexity focuses on CPU – a simulation on one supercomputer may have high computational complexity but low volume of data exchanged.

Use Case: **Characteristic:**	**Fast-Time Conceptual Trades**	**Fast-Time Detailed Evaluations**	**Real-Time Virtual Simulators**
Distributed Processing	Small number of low fidelity simulations	Potentially large heterogeneous cluster of simulations	Small number of high fidelity simulators augmented by medium fidelity simulations
Variability of Scenario Models and Data	High	High	Low – Medium
Complexity of input data **- Databases** **- Scenario Data**	 Low Low	 High High	 High Medium
Volume of Data Collected	Low	High	Medium (special case of audio/video)
Volume of Data Exchanged Between Simulations	Low	High	Medium
Total Computational Complexity	Low - Medium	High	Medium – High

Figure 2.5.1: Summary of Requirements for each Use Case

3 Simulation Architecture

3.1 Goals of an Architecture-based Approach

Having delineated the requirements of the three Use Cases, the next step involves determining in a general sense a system that satisfies those requirements, that is, creating a reference simulation architecture. Taking this approach satisfies several of the goals of the study: first, it facilitates determination of how many architectural variants are required to satisfy the Use Cases, and secondly, it provides a framework for evaluating the specific technologies examined in the study. This section describes the components of the architecture in general. Section 5 describes how the architecture serves each Use Case and how the technologies of Section 4 can be applied to creating implementations of the architecture.

Taking an architecture-based approach has benefits beyond the scope of the study and into system implementation. In general, implementing a system from an architecture reduces the effort in creating and using software systems. In particular, in a simulation community with multiple independent developers, architectural guidelines make reuse of models and infrastructure more straightforward, providing the ability to "plug and play" software components via standardized interfaces. The benefits of such an approach have been demonstrated within the Department of Defense, where the High Level Architecture (described further in Section 4.2.1) has been successfully employed to create large simulated worlds from independently developed simulations, where commonly accepted simulation frameworks (the function of a *simulation framework* is defined in Section 3.3) have facilitated infrastructure and model reuse and where common data models such as those of the High Level Architecture and the Synthetic Environment Data Representation and Interchange Specification (SEDRIS) [SE2001] have enabled interoperability.

An architecture can, in fact, form the basis of a simulation community. First, the composition enabled by an architecture creates a "greater good" which fosters community participation – that is, the opportunity for reuse and sharing of software and data encourages collaboration. Second, architecture allows planning across multiple projects for development of a *product line*. In this case, whereas no single project may have the resources to "do it all", project sponsors can coordinate to ensure that various projects in the aggregate create a greater whole. This type of allocation can be across technical lines (e.g., project A produces simulations, project B produces data collection tools) and/or domain lines (e.g., project A produces simulations of the en route airspace, while project B produces simulations of TRACON airspace). Again, the DOD experience is that communities develop around architectural and data standards[1].

[1] See, for example, the HLA SISO community (http://www.sisostds.org) and the SEDRIS community (http://www.sedris.org).

3.2 Development of a reference run-time architecture

It is important to point out that there can be many dimensions to an architecture.[2] For example, an architectural model can include:

- A *technical architecture*, comprising an abstract system specification consisting primarily of functional components described in terms of their behaviors and interfaces and component-component interconnections,
- A *domain architecture* containing a description of the domain elements being simulated and the information which is exchanged among those elements,
- A *software architecture* defining the software components needed to implement the functional elements of the technical architecture, and
- A *systems architecture* describing the systems solution (physical connection, location, and identification of computers, networks, etc) used to instantiate the technical architecture in order to simulate the elements represented in the domain architecture.

For the purposes of the study, a technical architecture is most appropriate form of reference architecture, with domain architecture included as an adjunct to technical architecture. A technical architecture allows specification of the functions that the components of the system must perform without getting into implementation details.

It should also be noted that an overall system may have a family of architectures. As shown in Figure 4.2.1, a the simulation lifecycle includes a number of phases. Each of these phases can have distinct architectures, from collaborative, human-centered environments at the front-end through scenario setup tools and simulation execution environments. The scope of this study is limited to run-time execution of simulations and so the reference architecture is primarily focused on the run-time aspects of simulation.

3.3 Description of the architecture

A run-time architecture for simulation must provide the functions required for the simulation to execute. For even the most basic standalone simulation this includes basic operations such as time advancement and flow of control within the software. For distributed simulations the set of functions required at run-time grows to include coordination and control of multiple independently executing simulation programs. For an analytical simulation the list of functions expands further to include capture and storage of data for use in assessment.

[2] For the many nuances of what is meant by architecture see, for example, http://www.sei.cmu.edu/architecture/definitions.html

The reference run-time architecture developed for this study, shown in Figure 3.3.1, provides these functions. The remainder of this section describes each of the components of the architecture. It is important before proceeding to note that in the following discussion the term *simulation* denotes an executable software program whereas *model* is used to denote a representation of some physical or behavioral aspect of the domain. As shown in the architecture diagram, a simulation contains a collection of models; for example, an airplane *simulation* may be composed of *models* of the dynamics of the airframe, the behavior of the pilot, etc. A *simulation system* is a collection of one or more simulations and tools designed to work together – an instance of the architecture.

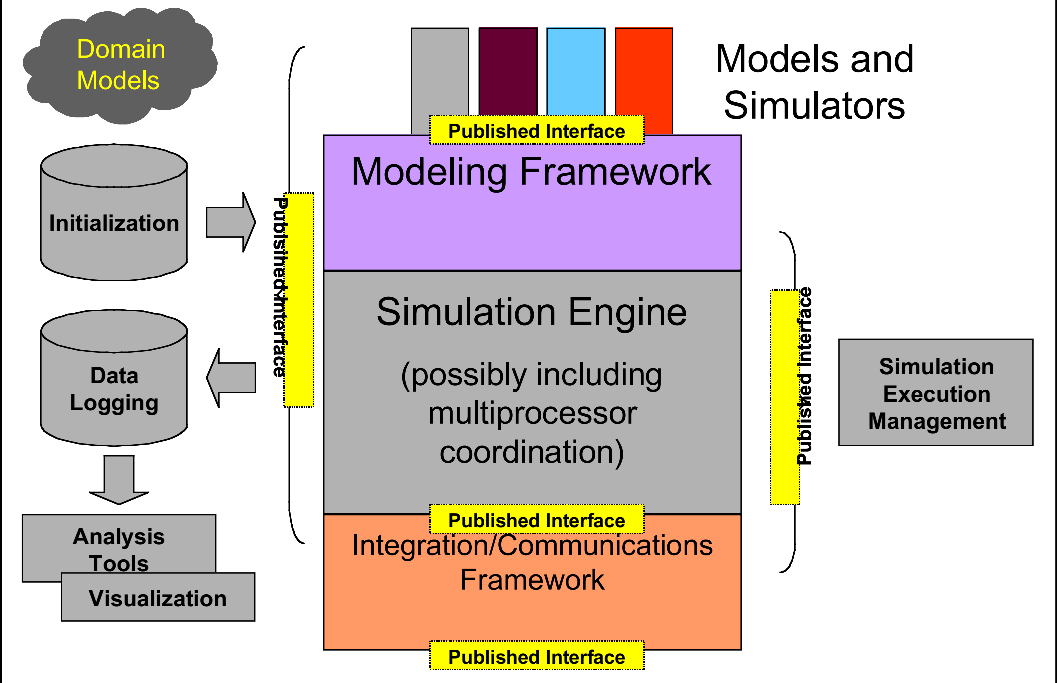


Figure 3.3.1: Reference Run-time Architecture

Within this architecture, utilities such as execution management tools, graphical displays, simulation gateways and the like exist at a peer level with simulations, making use of the same communications and coordination mechanisms as the simulations.

3.3.1 Simulation Execution Management

This function is shown in Figure 3.3.1 as Simulation Execution Management coupled with the Initialization and Data Logging databases.

Simulation Execution Management (SEM) covers control of the execution of the simulation system. The basic SEM operations are start/stop, pause/resume and save/restore. For a single simulation these can be command line or GUI-based functions, which control the execution of a single program. For a distributed simulation SEM is typically a distinct software application where start/stop must ensure coordinated startup of multiple programs, ensuring that simulation time doesn't start advancing until all participating simulations are fully initialized. Likewise, pause/resume must ensure that the various programs involved operate in a synchronized fashion to ensure consistency when resuming from a paused or saved state. Save/restore must ensure consistency of data as well.

Simulation Execution Management can also include multiple run control. Analytical simulations are used for *Monte Carlo* analyses meaning that they are run multiple times with stochastic variation of parameters. Analytical simulation studies may also involve multiple sets of Monte Carlo runs to assess sensitivities of output measures to input parameters (for example, capacity as a function of minimum in-trail spacing). A sophisticated Simulation Execution Management controller can automate the process of Monte Carlo simulation. Some execution management systems go as far as to automatically perform optimizations or backsolving against input parameters (see ASAC, Section 4.3.1, for an example of such a controller).

Simulation Execution Management also provides the linkage between run-time and the simulation phases which precede and follow it. It is SEM that invokes Simulation Initialization using data created in the pre-run-time scenario generation/simulation configuration phases and it is SEM that controls the collection of data to support post-run-time analysis and assessment.

Simulation Initialization is a part of Simulation Execution Management, which configures the simulation for execution of runs. At its most basic, Initialization allows a simulation to read parameters from some sort of configuration file generated during scenario generation/simulation configuration. For a distributed simulation this function can be much more complex. It can include distributing data to all of the simulations, which will be executing, assigning roles to each one (that is, controlling which domain elements will be simulated by each simulation program and computer), and configuring communications (network parameters) among the simulations. In Internet gaming where there is no central coordination among participants, simulation initialization typically includes verification that the various players are mutually compatible. In a related function, simulation initialization can distribute software to simulation computers to ensure that each has the appropriate versions of models.

Data Logging is the flip side of Simulation Initialization, capturing output data from the simulation system for use in downstream analyses. Data Logging in a distributed simulation environment must be able to merge multiple streams of data while preserving the temporal ordering. Consequently, Data Logging is controlled by SEM but it typically implemented in close coordination with the Simulation Engine.

SEM can also include fault tolerance features such as load balancing and automated transfer of simulated entities from one computer to another upon failure of a simulation computer.

3.3.2 Simulation Engine

This function is shown in Figure 3.3.1 as the Simulation Engine.

The Simulation Engine provides the functions necessary for the simulation to execute once it has been initialized. This component controls execution of the simulation's collection of infrastructure functions and models. This includes making sure that the temporal order of execution is maintained, even across the vagaries of multiple threads of control, networked communications and loosely synchronized computers.

The most basic simulation functions are time and event management. There are two basic types of simulations, those in which scenarios evolve based on increments of time and those, which are driven by events. Time-based simulations step time forward in increments and evaluate models at each new time. Discrete event simulations play out simulations as a sequence of events, where an event typically involves interactions between simulated entities and therefore invocation of models. The simulation engine advances the simulation by executing each event in turn. Events carry with them time tags and so time moves forward based on the time tags of the events.

Discrete event simulation differs from a time-stepped simulation in that only significant events are simulated. For example, if an airplane flight were simulated from takeoff to landing in a time-stepped simulation, the position of the plane would be updated at every cycle of the simulation. In a discrete event simulation, at the most general level there would be an event corresponding to the takeoff of the plane and an event corresponding to the landing of the plane. In between, it is assumed the plane flew its route. The plane's position can be queried at any point during the flight through further discrete events and events can be canceled if other objects affect the time-line of the plane. With proper tuning this scheme can result in much lower Central Processing Unit (CPU) use and much more efficient use of CPU resources than a time-stepped approach.

In distributed environments, synchronous execution of simulations becomes complex. Delays in passing time steps or event messages through the network can result in incorrect time ordering (for example, a pilot human behavior model may make a decision based on information available at a time however subsequently get a message which should have influenced the decision). A whole sub-field of the simulation domain, PADS (parallel and discrete event simulation) has evolved to study the optimal application of multi-processor environments to simulations.[3]

The simulation engine provides time/event management, invoking data distribution (see Communications Framework below) as required and managing process synchronization for multi-threaded or otherwise distributed simulations.

3.3.3 Modeling Framework

This function is shown in Figure 3.3.1 as the Modeling Framework.

Many simulation engines provide a set of tools that make it easier to develop models within the engine. Modeling Frameworks provide a layer of insulation above the raw time and event management features of the simulation engine allowing modelers to focus on representing the domain rather than on the computational aspects of simulation.

The Modeling Framework generally imposes a certain software model on the simulation developer. In many modern simulation engines, this model is object-oriented. Typically, the Modeling Framework provides a set of base classes which encapsulate simulation engine functions. Models must inherit from or use objects derived from these base

[3] See http://www.cs.utsa.edu/research/ParSim/ as an example.

classes. Implementing models as methods on such objects constrains the modeler but typically provides as a result transparent integration with the underlying engine's time, event and memory management. Object-oriented modeling frameworks typically represent the world as collections of aggregated objects, for example, modeling air operations as a collection of airplanes, where each airplane object comprises a set of model objects.

Other frameworks go beyond basic object-oriented concepts in an attempt to more closely match the modeling paradigm to the characteristics of the physical world. These agent-oriented modeling frameworks not only model the world via collections of aggregated objects but also impose constraints on the way objects interact with one another to more closely match real-world channels of communication.

In summary, the Modeling Framework makes it easier for modelers to use the capabilities of the simulation engine and, by providing standard interfaces, foster reuse of simulation models.

3.3.4 Models and Simulators

This function is shown in Figure 3.3.1 as the collection of Models and Simulators components.

Models are the pieces of software and/or hardware that represent portions of the NAS within the simulation system. For analytical simulation these are primarily blocks of software that model some physical or cognitive process. Examples include motion and fuel consumption models for aircraft, models of communications channels and human behavior models. For Human in the Loop simulation this set of software can also include interfaces to human role-players. Such interfaces may mimic real-world systems (such as in a cockpit simulator). Last, these blocks of code may include interfaces to hardware or wrapped legacy software to allow these elements to be incorporated into a simulation system.

Models and simulators are written to be compliant with the interfaces of a Modeling Framework/Simulation Engine. As a result, they are easily reused within that same framework, while migration to other frameworks requires adaptation of the model's algorithms to the interfaces of the new framework.

3.3.5 Integration/Communications Framework

This function is shown in Figure 3.3.1 as the Integration/Communications Framework.

The Communications Framework provides inter-process communications and coordination between the distinct applications and execution processes of the simulation system. Much as the Modeling Framework hides the details of the simulation engine from the modeler, the Communications Framework abstracts the details of inter-process communications from the simulation engine.

As previously described, maintaining coordination across multiple independently operating simulations or across parallel processors within a simulation is a complex task. It involves extending the time and event management functions of the simulation engine across multiple processes without allowing network performance to introduce perturbations into the simulation. The Communications Framework handles this coordination across multiple processors and computers, supplying integration functionality across multiple heterogeneous time and event management implementations.

As simulations grow in size, efficient utilization of networking resources becomes important. In a small simulation system it may be tolerable to broadcast information to the entire cluster of simulations and tools, however in large high performance systems such a practice may saturate the network or the simulation computers' ability to handle network traffic. For large simulation systems the Communications Framework provides efficient routing of data, delivering messages only to appropriate recipients. The Communication Framework may provide different delivery mechanisms (guaranteed, best-effort), different quality-of-service options, different topologies (client/server vs. peer-to-peer) and different data distribution mechanisms (such as publish/subscribe).

As part of implementing the above roles the Communications Framework abstracts from the simulation engine developer details of communications mechanisms such as CORBA or sockets, networking protocols such as TCP and UDP and network routing mechanisms such as multicasting.

3.3.6 Domain Models

This function is shown in Figure 3.3.1 as the Domain Models.

Domain models are not an active component of the simulation system however they are an important part of developing simulation systems. Those who have run afoul of simulation interoperability problems might interpret the Domain Models element in Figure 3.3.1 as a dark cloud hanging over simulation, however in fact this is the glue that holds simulations together.

Domain models are a representation of the area of the real world that is under study. A domain model represents the set of requirements that are common to systems within a product line, in this case, to all the components of a simulation system. The requirements represented in a domain model include:

- *Definition of scope for the domain*: the real-world elements that will be modeled within the simulation system (737-200, ADS-B system, traffic flow management decision-making, …)
- *Information or objects*: the way the real-world elements are decomposed within the simulation (for example, an engineering simulation may treat an airplane wing as a collection of many pieces, while an operational simulation may treat the whole aircraft as a single item).
- *Features or use cases*: The aspects or data associated with each object, and as a result with each model. As shown in Figure 3.3.6.1, depending on analysis needs a single entity type such as an airplane can be represented by many different sets of features such as physical models, cost models, etc.

- *Operational/behavioral characteristics*: The protocols involved with an domain object (for example, a pilot model upon receiving a message from a controller should acknowledge that message) and the behaviors associated with the object (aircraft update their position as they move but do not change their tail number).

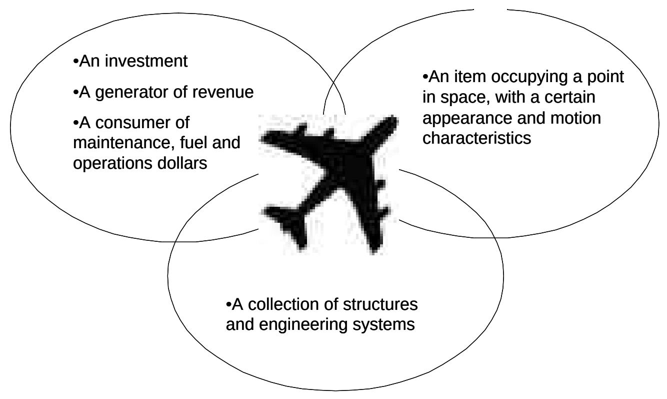

•An investment
•A generator of revenue
•A consumer of maintenance, fuel and operations dollars
•An item occupying a point in space, with a certain appearance and motion characteristics
•A collection of structures and engineering systems

Figure 3.3.6.1: Domain models representing several aspects of an aircraft

The most abstract form of a Domain Model is called a Conceptual Model (or Conceptual Model of the Mission Space in DOD parlance) and defines a common way of talking about the domain for a particular simulation community. Conceptual Models define the set of objects that could be simulated, including their interactions and hierarchical decomposition.

Reference data models are a more detailed Domain Model and describe the public attributes of each of the objects in a Conceptual Model and protocols for exchanging data and messages. Reference data models serve as a basis for simulation implementation and interoperability.

An Implementation Conceptual Model captures the previous models plus implementation information and is a simulation developer's way of translating modeling requirements (i.e., what is to be represented by the simulation) into a detailed design framework (i.e., how it is to be done), from which the software, hardware, networks (in the case of distributed simulation), and systems/equipment that will make up the simulation can be built. [Pa2001]

3.4 Legacy Software integration / reuse

Very few simulation developers have the luxury of developing entire systems from scratch. In most cases it is both expedient and desirable to incorporate existing models and data into new simulation systems. The structure of the architecture encourages both legacy integration and reuse of simulation components once developed. Software reuse can occur either through reuse of entire simulations as part of a new simulation system or by integrating existing models within new simulations.

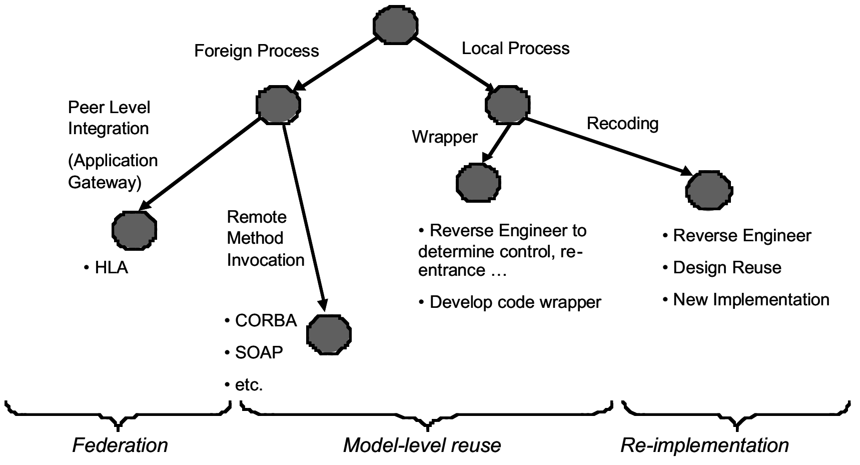
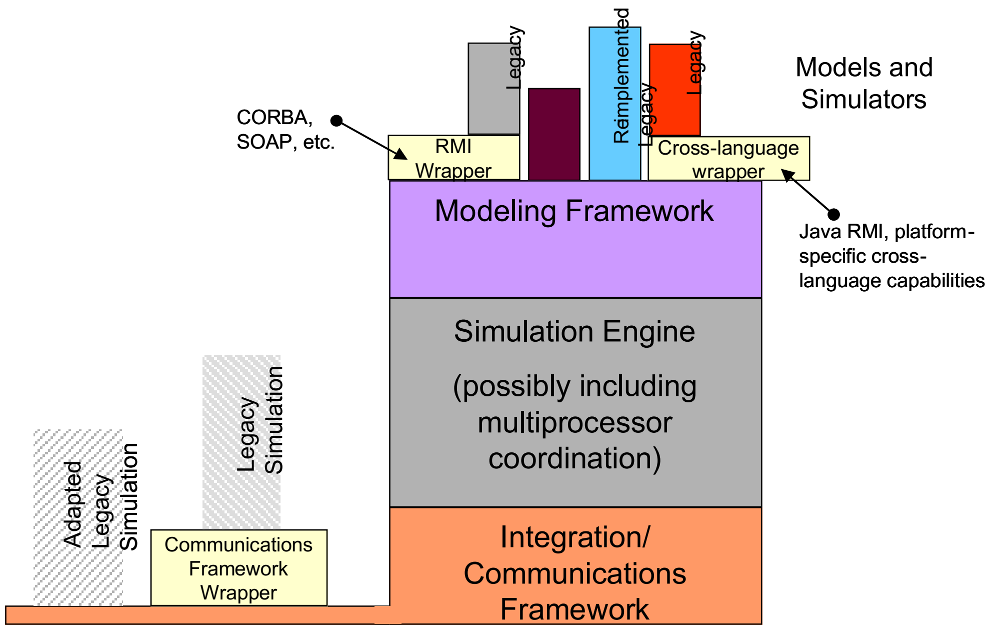


Figure 3.4.1: Mechanisms for software reuse in simulation

As shown in Figure 3.4.1[4], there are a number of paths to software reuse within the architecture. The potentially least intrusive way to integrate complete legacy simulations is via federation, or peer level integration of complete simulation applications through an application gateway or by direct integration using some sort of integrating protocol. Individual legacy models can be made accessible to a simulation by making the legacy code callable through a remote method invocation protocol such as CORBA or SOAP. Still tighter integration involves integrating legacy models with new simulations either by integrating existing code as part of the same executable or via reverse-engineering and re-



Legacy
Reimplemented Legacy
Legacy
Models and Simulators
CORBA, SOAP, etc.
RMI Wrapper
Cross-language wrapper
Java RMI, platform-specific cross-language capabilities
Modeling Framework
Simulation Engine
(possibly including multiprocessor coordination)
Integration/ Communications Framework
Adapted Legacy Simulation
Legacy Simulation
Communications Framework Wrapper

Figure 3.4.2: Reuse techniques within the reference architecture

[4] Adapted from [Or2001]

implementing algorithms into a compatible form. Figure 3.4.2 shows how these various approaches fit within the reference architecture.

Incorporation of legacy simulations through peer-level integration is shown in the figure through the Adapted Legacy Simulation and wrapped Legacy Simulations. There are a number of technical challenges involved in making independently developed legacy simulations interoperate with one another. Simulations that were not specifically designed for inter-operation typically use conflicting representations of the same elements of the problem domain, or of the environment, or of the passage of time. Different simulations may model overlapping aspects of the domain, making integration difficult because one simulation's representation of one of these overlapping elements needs to be "turned off," and that simulation must be altered to accept input from the corresponding element represented in another simulation. All of these differences between simulations need to be rationalized to create a useful analytic system. The two cases shown in the figure differ in whether the reconciliation of conflicting control and modeling structures across simulations has been resolved by adapting the legacy simulation or whether a intermediary piece of software has been developed to serve as a gateway between the new simulation system and the legacy simulation. The latter approach can impose significant restrictions in both system performance and representational consistency but often requires less effort than modifying (and potentially re-validating) existing software. The DOD High Level Architecture (see Section 4.2.1) was developed to support these types of integration and has been used successfully with both of the approaches described above.

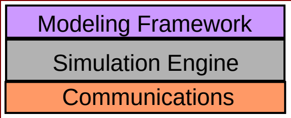
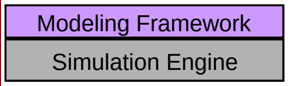
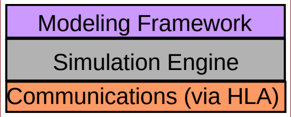
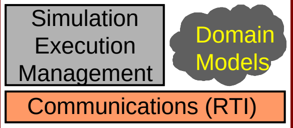
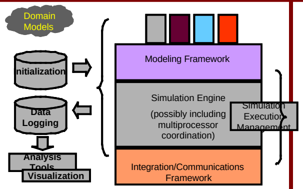
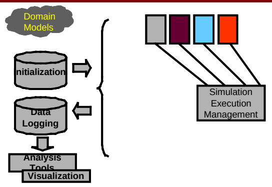
The more tightly coupled approaches to model integration are shown in Figure 4.4.2 as different ways of integrating models with a modeling framework/simulation engine. These include a Remote Method Invocation (RMI) wrapper around the model, a cross-language wrapper for integration within a single software executable and re-implementation of models in a new simulation.

3.4.1 Planning for Reuse

There is an overall connection among development of a system from a common architecture (Section 3.1), adoption of domain models (Section 3.3.6) and reuse. There is no way to turn back time and modify the original development of legacy models and so they must be taken for what they are, however it is possible to facilitate reuse going forward. The development of models and simulations in conformance with architectural and data standards makes it far easier to reuse these pieces of software in future analyses or systems. The investment involved in developing domain models and in designing simulations so that they can be federated is rapidly recouped in reductions in development effort for subsequent similar systems.

4 Technology Reviews

This section provides an overview and evaluation of each technology surveyed as part of this study. Each technology is also mapped into components of the reference architecture. The various candidate technologies are grouped by sub-section according to its primary role in the architecture. The results of this section are summarized in Figure 4.1.

Technology	Architectural Mapping	Applicability to Use Cases*	Comments	HLA
SPEEDES		FTCT: Applicable but overkill FTDE: Applicable RT: Not Applicable	Powerful multi-processing simulation engine, but reaping the benefits of parallelism takes effort	Yes
SWARM		FTCT: Possibly Applicable FTDE: Not Applicable RT: Not Applicable	Sim engine from artificial life community. Insufficient scalability, power for NAS applications. Agent modeling paradigm is a plus.	No
RFS	 Modeling Framework Simulation Engine Communications (via HLA)	FTCT: Applicable FTDE: Applicable, possible performance limitations RT: Applicable	NAS domain simulation engine. Extensible model collection. Lack of releasable, "shrink wrap" version	Yes
HLA	 Simulation Execution Management Domain Models Communications (RTI)	FTCT: Applicable FTDE: Applicable RT: Applicable	Mature processes, tools and software for integrating/composing independent simulations.	Yes
AIRMM	 Domain Models nitialization Data Logging Analysis Tools Visualization Modeling Framework Simulation Engine (possibly including multiprocessor coordination) Simulation Execution Management Integration/Communications Framework	FTCT: Concept Applicable FTDE: Applicable RT: Applicable	Wide-ranging set of architectures for full lifecycle of simulation, from knowledge management through simulation execution and analysis	Yes
ASAC	 Domain Models nitialization Data Logging Analysis Tools Visualization Simulation Execution Management	FTCT: Concept Applicable FTDE: Not Applicable RT: Not Applicable	Good conceptual architecture for FTCT and analytical simulation execution management.	No

**FTCT = Fast-time Conceptual Trades, FTDE = Fast-time Detailed Evaluation, RT = Real-time*

Figure 4.1 Technology Capability Matrix (Page 1 of 2)

Technology	Architectural Mapping	Applicability to Use Cases*	Comments	HLA
TRANSIMS	Modeling Framework Simulation Engine Communications	FTCT: Applicable FTDE: Applicable RT: Not Applicable	Simulation environment for multi-modal ground transportation problems. Composable. Parallel simulation engine.	No
DESIREE	Modeling Framework Simulation Engine	FTCT: Not Applicable FTDE: Not Applicable RT: Applicable	Simulation engine for ground portion of air-traffic control systems. Focus on human factors.	No
NIAC	Domain Models	FTCT: Applicable FTDE: Applicable RT: Applicable	Domain modeling effort for development of ATM systems. Some reuse for NAS simulation development (e.g., HLA FOMs)	N/A
Osim, Stage, VR-Link	Modeling Framework Simulation Engine Communications	FTCT: Not Applicable FTDE: Not Applicable RT: Applicable	COTS Simulation engines. Specialized to real-time, except for VR-Link.	Osim, VR-Link: Yes

Figure 4.1 Technology Capability Matrix (Page 2 of 2)

**FTCT = Fast-time Conceptual Trades, FTDE = Fast-time Detailed Evaluation, RT = Real-time*

4.1 Simulation/Modeling Engines

4.1.1 Synchronous Parallel Environment for Emulation and Discrete Event Simulation (SPEEDES)

4.1.1.1 Architectural Category

The Synchronous Parallel Environment for Emulation and Discrete Event Simulation (SPEEDES) is a general-purpose discrete event distributed simulation engine and modeling framework, focused on scalability through parallel execution.

Architectural Elements: Modeling Framework, Simulation Engine (parallel discrete event), Communications (multi-processor and HLA).

4.1.1.2 Characteristics

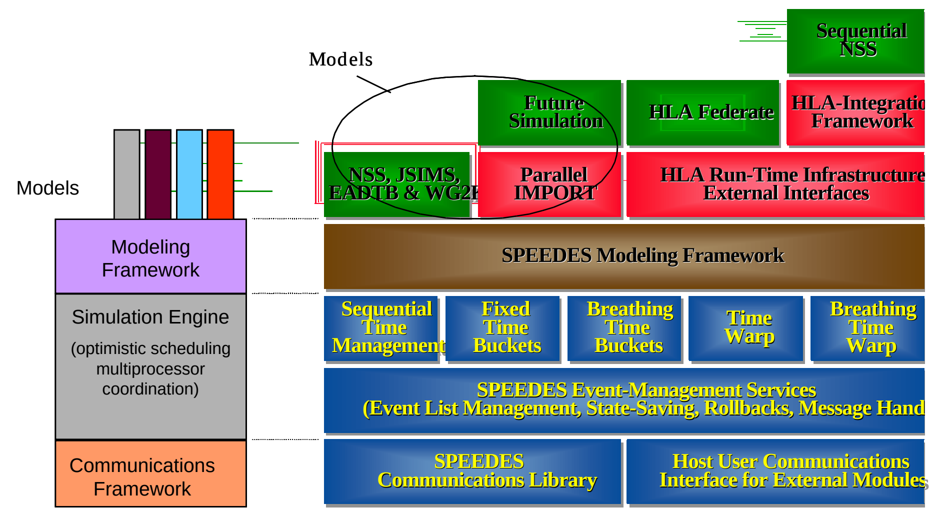
Synchronous Parallel Environment for Emulation and Discrete Event Simulation (SPEEDES)	
History	Developed by JPL 1990-1995, Metron since 1996
User Community	Joint Simulation System (JSIMS), Wargame 2000, Joint National Test Facility (JNTF), Naval Simulation System (NSS), Extended Air Defense Test Bed (EADTB). These efforts involve numerous organizations including: TRW, SAIC, Raytheon, Hughes Aircraft, MITRE, Sparta, Logicon, OriginalSim, System Integration Software (SIS), EcoSoftware, Jet Propulsion Laboratory (JPL), Los Alamos National Laboratory (LANL), Dartmouth College, University of Virginia, and others.
Cost	Free to NASA
Community Web Sites	www.speedes.org, www.speedes.com
Documentation	Extensive user manual, API reference and examples, books, papers
Software	C++, available on Unix and Windows NT. Integrated with HLA
Sources of information:	[St1999], [SP2000], [Hi2001], Conversations with Dr. Edward Powell, former JSIMS simulation architect.

4.1.1.3 Summary

SPEEDES is a discrete event simulation engine and modeling framework. Development of SPEEDES has emphasized distributed processing, focusing on integration of objects distributed across large numbers of processors. One of the key features of SPEEDES is its ability to preserve causally correct event processing in a parallel processing environment.

SPEEDES is first and foremost a discrete event simulation engine. SPEEDES runs discrete event simulations in parallel by distributing the simulation objects to different processors. Each processor is then responsible for executing all events on the simulation objects that it was assigned, though any event may affect objects on other processors.

The SPEEDES architecture, shown in Figure 4.1.1.1 provides communications, event, and time management, and a modeling framework. The SPEEDES modeling framework, shown in Figure 4.1.1.2 provides the basic functions needed for event-based simulation – objects, various types of events, event messages and event handling. SPEEDES provides



Sequential NSS
Models
Future Simulation
HLA Federate
HLA-Integratio Framework
Models
NSS, JSIMS, EADTB & WG2E
Parallel IMPORT
HLA Run-Time Infrastructure External Interfaces
Modeling Framework
SPEEDES Modeling Framework
Simulation Engine
(optimistic scheduling multiprocessor coordination)
Sequential Time Management
Fixed Time Buckets
Breathing Time Buckets
Time Warp
Breathing Time Warp
SPEEDES Event-Management Services
(Event List Management, State-Saving, Rollbacks, Message Hand
Communications Framework
SPEEDES Communications Library
Host User Communications Interface for External Modules

Figure 4.1.1.1: SPEEDES Architecture

functions specific to optimistic parallel simulation including checkpoint, rollbacks and memory management as well as tools for optimizing the allocation of simulation object processing across multiple processors. SPEEDES also provides some diagnostic tools including event tracing and event usage statistics.

SPEEDES provides HLA interoperability in several ways. First, SPEEDES is an HLA client, allowing SPEEDES-based simulations to interoperate with other HLA simulations via standard RTI mechanisms. Second, SPEEDES has implemented the HLA RTI interface so that the SPEEDES kernel itself can serve as an RTI. Under this scheme, multiple SPEEDES and/or non-SPEEDES federates operating on high-performance computing platforms can interoperate via the standard RTI interface, with RTI communications being implemented using high-speed shared memory mechanisms. Third, SPEEDES can serve as a gateway allowing remote federates to interoperate with SPEEDES-based simulations without employing separate RTI software.

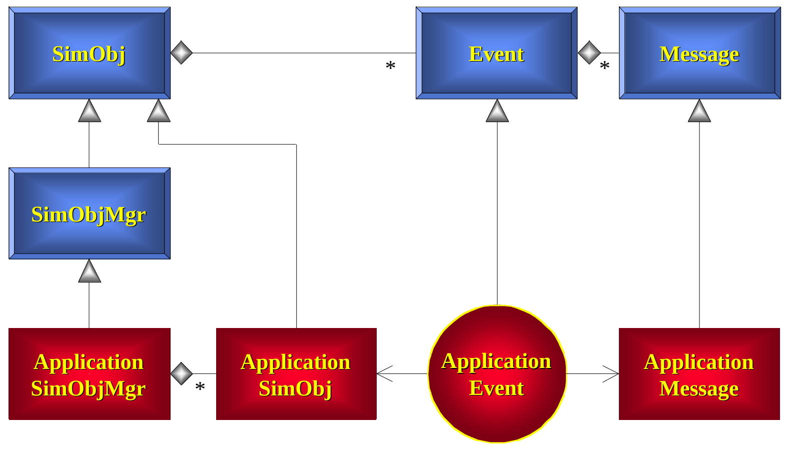


Figure 4.1.1.2: SPEEDES Modeling Framework

4.1.1.4 Evaluation

It should be noted that parallel discrete event simulation (PDES) is a specialized art and that employment of a PDES engine does not guarantee improved performance. A 1993 article by Professor Richard Fujimoto, himself a PDES practitioner, questioned whether the PDES field would survive, given the difficulty of applying parallel simulation techniques. [Fu1993] A recent Air Force prototyping experiment actually showed a decrease in performance under SPEEDES as the size of the parallel cluster increased [Hi2001]. Such caveats notwithstanding, parallel computing remains a practical way to tackle many classes of large simulation problems. Fujimoto's article concludes that the field will survive for this reason, and the Air Force project chose to continue with parallel simulation despite initially disappointing results.

SPEEDES is a mature simulation engine that has been used in a number of large projects across a variety of hardware architectures ranging from massively parallel machines to distributed networks of fast workstations. It could be a good choice as a framework for NAS simulations, subject only to caveats about the complexity of programming for and optimizing the execution of PDES.

Applicability to Use Cases:

- Fast-time Conceptual Trades – Applicable but probably overkill
- Fast-time Detailed Evaluation – Applicable
- Real-time Virtual Simulations – Not Applicable

4.1.2 Agent-Based Simulation/SWARM

4.1.2.1 Architectural Category

Swarm is a simulation engine and modeling framework focusing on an *agent-based simulation* paradigm. The product's origins are in the field of Artificial life, an approach to studying biological systems focusing on discovery of macro level mechanisms and patterns by observing the collective interactions of micro-level models.

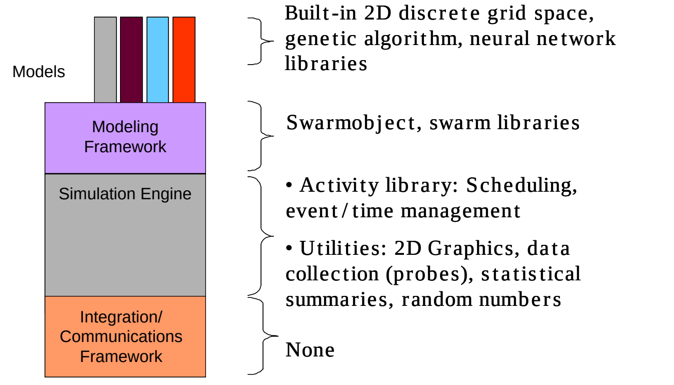
Architectural Elements: Modeling Framework, Simulation Engine

4.1.2.2 Characteristics

SWARM	
History	Developed at Sante Fe Institute starting in 1994
User Community	User Community: Primarily academia, with projects in biology, ecology, economics, political science, computer science, anthropology, geography and defense (see http://www.swarm.org/community-links.html)
Cost	Free under Gnu licensing terms
Community Web Sites	Swarm development group (swarm.org), swarm.com
Documentation	User manual, API reference and examples, books, papers
Software	Objective C, with interfaces to Java, Scheme, XML. Available for Windows and Unix. Supporting graphics, data collection libraries
Sources of information:	[Da1999], [SD2000]

4.1.2.3 Summary

Swarm is an agent simulation framework originally inspired by artificial life concepts. Thus, its modeling paradigm focuses on applications with large numbers of interacting entities, or agents. Agents within Swarm can be organized hierarchically into collections called *swarms*.



Models
Modeling Framework
Simulation Engine
Integration/ Communications Framework
Built-in 2D discrete grid space, genetic algorithm, neural network libraries
Swarmobject, swarm libraries
• Activity library: Scheduling, event/time management
• Utilities: 2D Graphics, data collection (probes), statistical summaries, random numbers
None

Figure 4.1.2.1: SWARM libraries mapped to the reference architecture

The Swarm engine offers a fairly basic set of capabilities. Like most simulation frameworks, it offers time and event management. Each Swarm agent maintains one or more *schedules*, or lists of actions. Each action is in effect an event which triggers a method call, invoking some pre-defined behavior of the agent. Actions can be invoked based on time (from a schedule) or based on interactions with other agents. Swarm is a fast-time time-stepped model with a variable time step. The Swarm engine does not contain any features for distributed computing.

Swarm contains several features to support model development, observation and analysis. The first is the Observer swarm, which is typically a parent of the other swarms in a simulation. Observer objects can input data into model swarms (setting simulation parameters, for instance) and read data out of the model swarm (collecting statistics of the behavior of agents). Just as with model swarms, an observer swarm has a collection of objects (the instrumentation), a schedule of activity, and a set of inputs and outputs. The activity of the observer schedule is to drive data collection – e.g., read a number out of the model, draw it on a graph. The inputs to the observer swarm are configurations of the observer tools: what sorts of graphs to generate, for instance. The outputs are the observations.

Swarm also contains a representation of physical space with a rectangular coordinate system, data probes for connection of model swarms to observer swarms, random number generation and a 2D graphics library.

4.1.2.4 Evaluation

Swarm was created as a simple way for non-programmer scientists to create virtual experiments – to in effect create a petri dish on the computer. As such it is long on simplicity of interface (the 1,200 line "simple" *heatbugs* example application aside) and short on fancy computational characteristics. Swarm's native language is Objective C, though it can run models written in Java and Scheme with declarations in XML. The Swarm community is primarily centered in academia.

Swarm presents a novel set of tools for looking at an interesting class of problems, however its lack of scalability features, absence of features for integration with other simulations, somewhat limited language support and lack of proven industrial-scale use lead to a recommendation that it not be considered as a primary simulation engine for NAS simulations.

Applicability to Use Cases:

- Fast-time Conceptual Trades – Possibly Applicable
- Fast-time Detailed Evaluation – Not Applicable
- Real-time Virtual Simulations – Not Applicable

4.1.3 Georgia Tech Reconfigurable Flight Simulator (RFS)

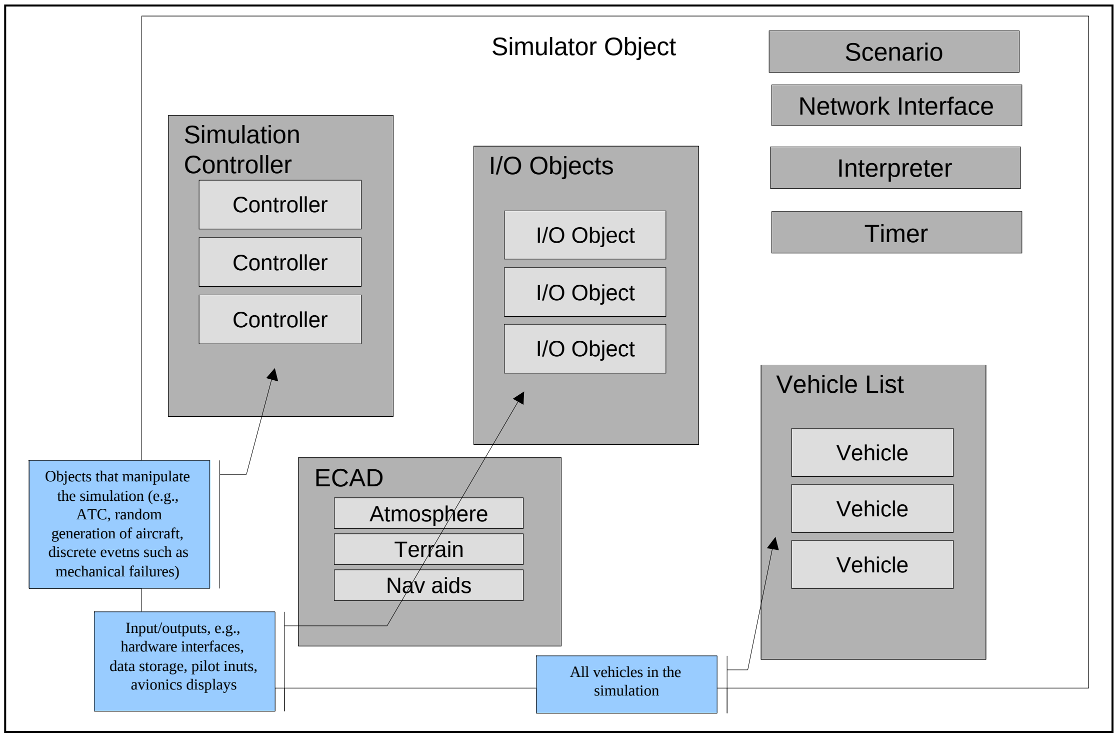
4.1.3.1 Architectural Category

The Georgia Tech Reconfigurable Flight Simulator is described in the literature as a simulation engine and modeling framework, focused on the particular needs of the analysis and design communities. The modeling framework uses an object-oriented paradigm for representation of simulation objects. RFS has also been used as a framework for a gateway to interconnect multiple simulations.

Architectural Elements: Modeling Framework, Simulation, Communications (via HLA RTI).

4.1.3.2 Characteristics

Georgia Tech Reconfigurable Flight Simulator (RFS)	
History	Developed at Georgia Tech over the last several years, funded by NASA.
User Community	Small, primarily GA Tech, NASA Langley
Cost	Free to NASA
Community Web Sites	None
Documentation	Several papers. Status of programmer documentation is unknown.
Software	C++, with OpenGL graphics. Integrated with HLA.
Sources of information:	[Ip2000], [Pr2000],[Pr2001], conversation with Dr. Amy Pritchett of Georgia Tech


Simulator Object
Scenario
Network Interface
Interpreter
Timer
Simulation Controller
Controller
Controller
Controller
I/O Objects
I/O Object
I/O Object
I/O Object
Vehicle List
Vehicle
Vehicle
Vehicle
ECAD
Atmosphere
Terrain
Nav aids
Objects that manipulate the simulation (e.g., ATC, random generation of aircraft, discrete evetns such as mechanical failures)
Input/outputs, e.g., hardware interfaces, data storage, pilot inuts, avionics displays
All vehicles in the simulation

Fiaure 4.1.3.1: Georaia Tech Reconfiaurable Fliaht Simulator Architecture

4.1.3.3 Summary

The Reconfigurable Flight Simulator was designed to meet the requirements of a simulator useful for aerospace research and design activities. The goal of the system is to provide a generic framework within which aerospace researchers can combine models to create simulations.

The RFS has the goal of satisfying a defined set of simulation requirements, including:

- *Flexibility*: the simulation should not be fundamentally constrained to a specific set of models, mode of operation or fidelity level
- *Accessibility*: The simulation should be accessible to the general user, that is, a technical user who is not a simulation specialist
- *Robustness*: The simulation should support modification without introduction of unpredictable side effects due to complex model inter-dependencies
- *Extensibility*: Addition of new components should be possible without fundamental changes to the system architecture

To satisfy these requirements the RFS has developed a simulation engine with a single processor kernel and an object-oriented modeling framework. The engine consists of a set of core simulator objects that handle object creation/destruction and list management, time management, coordinates and environment. These objects are augmented by a domain-specific modeling framework, as shown in Figure 4.1.3.1.

The RFS modeling framework contains three major categories of objects: first, there are vehicles, which include moving elements of the simulation including aircraft and ground vehicles. Second, there is an explicit base class for linking in Input/Output (I/O) objects such as hardware interfaces, data storage components, pilot interfaces and avionics displays. Third, there is a Controller/Measurement category to represent objects that manipulate, control or measure aspects of the simulation. Examples of this latter category include ATC controllers, random aircraft generation and discrete events such as mechanical failures.

The RFS engine has some unique features for supporting visibility into analyses. In particular, the Object Data/Method Extensions (OD/ME) capability allows models to declare and invoke interfaces at run-time. This capability allows flexibility in model integration and in integration with external software controllers (including command line control providing analysts with an *ad hoc* interface to simulations at run-time). The disadvantage of this interface is that all OD/ME calls run through an interpreter and so are considerably slower than native function calls.

RFS has been used in both fast-time and real-time applications. In real-time mode thee framework has also been used to tie together event-driven models and time-stepped human factors models (in particular, MIDAS), creating a hybrid representational paradigm.

4.1.3.4 Evaluation

RFS was somewhat hard to evaluate, given the small amount of published information available. This speaks to its newness on the scene and lack of packaging as a "shrink-wrap" tool as well as the small size of its user community.

The framework appears to be useful for small to medium size simulations, both for real-time and fast-time applications. Its lack of multi-processor capabilities and interpreted model interface language may point to limitations in potential scalability. On the other hand, its implementation of HLA for both external communications and integration of multiple RFS instances (including distribution of the OD/ME protocol via HLA) does provide an opportunity for scale and integration. The flexible software interface strategy of RFS could make it a good candidate for quick turn-around simulations and for interconnection with virtual simulators.

Applicability to Use Cases:
- Fast-time Conceptual Trades – Applicable
- Fast-time Detailed Evaluation – Applicable, possibly with performance limitations.
- Real-time Virtual Simulations – Applicable

4.1.4 TRANSIMS

This software was not among those specifically called out by NASA and so only a brief survey is presented.

Architectural Elements: Modeling Framework, Simulation Engine (parallel), Communications (multi-processor).

Transportation Analysis and Simulation System (TRANSIMS) is an integrated system of travel forecasting models designed to give transportation planners information on traffic impacts, congestion, and pollution. TRANSIMS models create a virtual metropolitan region with a complete representation of the region's individuals, their activities, and the transportation infrastructure. TRANSIMS was originally developed for the Department of Transportation at Los Alamos National Laboratory. As of October of 2000, development of a releasable version transitioned to an outside contractor. [TR2000]

TRANSIMS is primarily a *node and link* model of multi-modal ground transportation, however it provides some simulation engine features and simulation support tools that are of more general interest. First, while TRANSIMS is not designed to interoperate with other simulations it can itself be distributed in a cluster-computing environment (specifically a "Rockhopper" cluster).

Second, TRANSIMS has created an environment where different simulation modules, for example, of population generation, route planning, traffic simulation and tailpipe emissions calculation, can be chained together in flexible ways for varying studies. This mirrors the ASAC concept of dynamic assembly of models based on open APIs.

Third, like ASAC, TRANSIMS has sets of simulation control and data manipulation tools for performing multiple iteration of runs for a study or experiment, and for filtering sorting, indexing cataloging output data as well as “noising” (introducing stochastic variation) input data files.

TRANSIMS services a different domain than NAS simulation. Some of the problems are similar, however some are quite different (for example, individual travelers’ choices in mode of surface travel – be it car, bus, or carpool – do not come into play in the ATM domain). TRANSIMS also suffers from being a closed system – it does not integrate with other simulations. However, given the level of investment (> $25M, per the TRANSIMS web site), project scalability features (current focus is on a scenario with 120,000 links and 1.5 million travelers -- an order of magnitude larger than the achieved Dallas/Fort Worth simulation of 10,000 links and 200,000 travelers) it may be worth investigating as a simulation engine. [TR2001][TR2001b] Availability to NASA is unknown.

4.1.5 DESIREE

This software was not among those specifically called out by NASA and so only a brief survey is presented.

Architectural Elements: Modeling Framework, Simulation Engine.

The Distributed Environment for Simulation, Rapid Engineering and Experimentation (DESIREE) was developed by the FAA William Hughes Technical Center R&D Human Factors Laboratory. DESIREE is a simulation engine designed as a tool for building simulations of the ground portion of Air-Traffic Control systems. The focal point of DESIREE lies in user interface simulation and so its modeling framework is designed for rapid reconfigurability of user interface elements. Models connect into the DESIREE infrastructure via a publish/subscribe mechanism. This concept, which mirrors both common notions of User Interface callbacks, allows model-level components to be independently developed and fairly simply interconnected at run-time.

DESIREE includes tools which support its application goals, included integrated Q&A (a facility for interrupting the simulation and asking the user a question), event scripting and data collection/analysis tools.

DESIREE is a stand-alone real-time framework developed for human factors analysis, and as such is not an optimal candidate as a generic simulation framework for NAS simulation. It does, however, include some concepts that could be useful in developing generic, rapidly reconfigurable simulations and model libraries.

4.1.6 COTS Frameworks

This software was not among those specifically called out by NASA and so only a brief survey is presented.

Architectural Elements: Modeling Framework, Simulation Engine, Communications (HLA for VR-Link).

In addition to review of relevant government efforts, the study also included a look at the state of Commercial Off the Shelf (COTS) simulation engines. A survey of the Defense Modeling and Simulation Office web site, www.dmso.mil, led to three representative frameworks: OSim, from OriginalSim [OS1998][OS1998b], STAGE, from Virtual

Prototypes, Inc. [Be1997] and VR-Link, from MaK Technologies. Evaluation of these tools is useful in that it provides a perspective into both the requirements for and the tools available for creating a NASwide simulation environment. Because of the focus within this study on architecture rather than tools, COTS simulation libraries such as Simscript were not included in the survey.

All three of the tools evaluated were focused on real-time applications. OSim takes a somewhat architecture-driven approach to simulation. The product includes a simulation engine with defined interfaces for integrating models. The product includes some tools for managing libraries of models. It also includes tools for both data visualization and 2D/3D view creation. Osim includes HLA integration.

STAGE is a real-time simulation engine designed for training environments. Simulated entities can be scripted in STAGE's own scripting language. The product has defined APIs that allow users to insert additional models into the system. The tool provides interoperability through the older Defense DIS protocol, but not via HLA.

VR-Link is a venerable simulation toolkit from the DIS/HLA world. It is in a sense the most barebones of the three, offering only some basic simulation engine and network tools, consisting in part of wrappers around the HLA RTI. VR-Link also integrates with other MaK products which provide viewing and data collection via HLA.

4.2 Simulation Interoperability/Integration

4.2.1 DOD High Level Architecture (HLA)

4.2.1.1 Architectural Category

The Department of Defense (DOD) High Level Architecture (HLA) is a set of processes, specifications, tools and software designed to facilitate integration and management of independently developed simulation applications. The HLA Runtime Infrastructure (RTI) is a communications framework that allows simulation engines to coordinate and synchronize their time, event and communications management functions.

Architectural Elements: Simulation Execution Management, Domain Modeling, Communications.

4.2.1.2 Characteristics

High Level Architecture (HLA)	
History	Developed by DOD starting in 1995 based on earlier distributed simulation experience
User Community	Large DOD User base. Sponsored by Defense Modeling and Simulation Office (DMSO).
Cost	Free to NASA
Community Web Sites	hla.dmso.mil, www.sisostds.org
Documentation	API reference and examples, books, papers
Software	C++, available on multiple platforms with interfaces to multiple languages (CORBA IDL, Ada95, Java).

Sources of information:	[DM1998][DM1999]

4.2.1.3 Summary

<u>DOD M&S Integration Experience</u>

The DOD Modeling and Simulation (M&S) community has in the past ten years focused a great deal of attention and resources on the problem of integrating heterogeneous models and simulations. The DOD simulation community was at one time oriented almost entirely to producing stovepipe solutions to M&S problems. Numerous incompatible simulations existed within the DOD domain. Meanwhile, the DOD was facing increasingly demanding operational requirements, including more complex missions in a vastly expanded mission space; increased complexity of missions, plans, and systems; and a much more restrained fiscal environment. Advanced M&S capability was believed to hold a key to more cost effective analysis, design, operational planning, doctrine development, and training. An intellectual consensus for interoperability consequently developed in the early 1990s, and "opening up" and linking existing simulations became the primary technical challenge of the DOD M&S community.

Various integration mechanisms were proposed and implemented. First, a centralized server-based integration approach was constructed in the Aggregate-Level Simulation Protocol (ALSP) program, with significant functionality, but little generality, openness, and scalability. A peer-to-peer message-passing architecture called Distributed Interactive Simulation (DIS) was created, in which standard message types and protocols were agreed to by a large fraction of the DOD M&S community. DIS was achieved widespread acceptance but suffered from lack of flexibility in defining messages as well as scalability and performance problems. In 1994, leaders in the DOD simulation community sponsored the Advanced Distributed Simulation Architecture Study to address the general needs of the DOD M&S community and build upon the strengths of ALSP and DIS. This study resulted in the definition of the DOD High Level Architecture (HLA) for Modeling and Simulation in 1995, and its subsequent standardization in 1999. Today, the majority of DOD simulations are HLA-compliant or are moving in that direction.

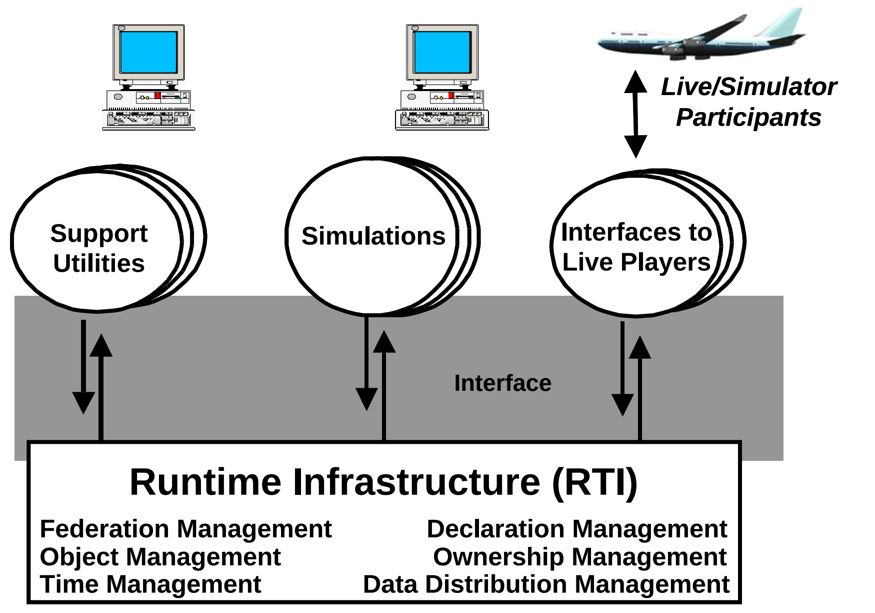


Figure 4.2.1.1: High Level Architecture Runtime Infrastructure

<u>HLA Elements and Terminology</u>

The HLA provides a blueprint for integration of existing simulations and provides guidelines for how to create future simulations with interoperability in mind. The HLA specifies a set of rules, an interface specification to a communications infrastructure called the Run-Time Infrastructure (RTI), and a format called the Object Model Template (OMT) for use in defining message types for information interchange. The RTI provides the key set of services required of a distributed simulation system. Figure 4.2.1.1 shows how the RTI connects simulations

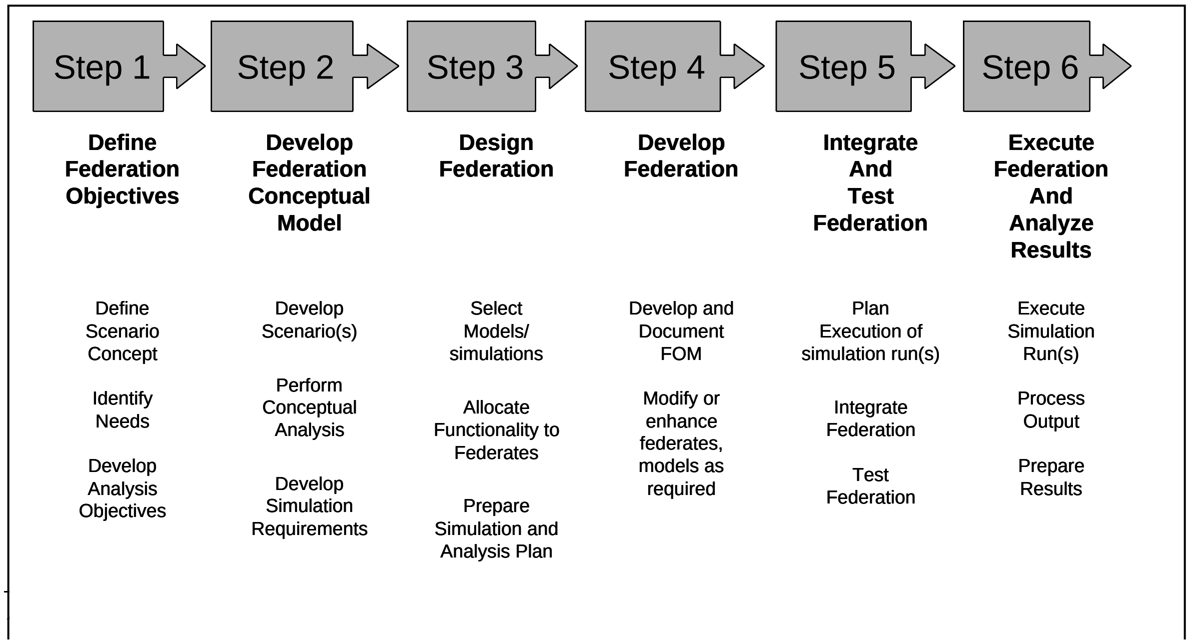
A set of simulations (previously referred to in this report as a “simulation system”), integrated using an HLA RTI and a specific object model, is called a “federation,” with the specific object model being used in a given federation called the “federation object model” (FOM). Each simulation within the federation is referred to as a “federate”.

Besides its core architectural elements, the HLA also provides auxiliary information, including:

- A standard process for integrating simulations called the Federation Development and Execution Process (FEDEP),
- An available set of COTS and GOTS tools for common setup, monitoring, data collection and analysis functions,
- A standard representation of weather and other aspects of the environment called the Synthetic Environment Data Representation and Interchange Specification (SEDRIS),
- A GOTS implementation of the RTI, that operates on a wide range of hardware platforms and operating systems, including virtually all commonly used PC and UNIX systems.
- Guidance on how to use the RTI in demanding, high performance applications.

<u>Federation Development and Execution Process (FEDEP)</u>

The FEDEP is a high level process which facilitates integration of independent simulations and fosters reuse at the simulation level by creating a composable simulation environment wherein independent simulation modules can be combined as needed based


Step 1
Step 2
Step 3
Step 4
Step 5
Step 6
Define Federation Objectives
Develop Federation Conceptual Model
Design Federation
Develop Federation
Integrate And Test Federation
Execute Federation And Analyze Results
Define Scenario Concept
Identify Needs
Develop Analysis Objectives
Develop Scenario(s)
Perform Conceptual Analysis
Develop Simulation Requirements
Select Models/ simulations
Allocate Functionality to Federates
Prepare Simulation and Analysis Plan
Develop and Document FOM
Modify or enhance federates, models as required
Plan Execution of simulation run(s)
Integrate Federation
Test Federation
Execute Simulation Run(s)
Process Output
Prepare Results

on user needs.

The first part of the FEDEP process is to understand the analysis objectives that the federation will support. A federation may be put together for a single exercise or analysis or it may persist over time. The top-level conceptual model of the federation is used as a roadmap for actually instantiating the federation.

The second step in the FEDEP process is to perform detailed design of the federation by selecting the models or simulations that will be included and deciding what functionality each model and simulation support/management tool brings for the federation. In many cases, there may be an overlap in functionality between federates; however, the designers of a federation specify which federates provide which functionality for the federation.

The most important part of the process of establishing interoperability between the various simulations is the development and documentation of the Federation Object Model used for that federation. The FOM includes a detailed description of all of the information exchanged in distributed federation and needs to take into account the capabilities of the various members of the federation. Sometimes, when domain representations used in different simulations are similar, the FOM development process is straightforward. Other times, when legacy simulations are used and representations are very dissimilar, a more extensive process of negotiation, translation, and alteration is required.

Figure 4.2.1.2 shows the FEDEP.

The HLA also defines several other related object models. The Simulation Object Model (or SOM), defines all the objects, attributes and interactions that a particular simulation is capable of using externally (not all of which may be used in a particular FOM). The Management Object Model (MOM) identifies a set of objects and interactions used to manage a federation.

Runtime Infrastructure (RTI)

The RTI software provides a set of services used by federates to coordinate their operations and exchange data during run-time execution. Access to RTI services is defined by the HLA Interface Specification, and all data exchanged through the RTI is defined by the Federation’s FOM and the MOM. HLA RTI services are summarized in the RTI box in Figure 4.2.1.1.

At run-time each federate may publish and subscribe to information (“objects” and “interactions” in HLA parlance) based on the total set of distributed information described in the FOM. The RTI handles distribution of information from publishers to subscribers. The RTI contains several optimization schemes that ensure that each simulation gets only that information to which it subscribes, supporting scalability by reducing network and processor loading needed to filter extraneous data.

The RTI also contains mechanisms for coordinating time across simulations, from loose time coordination useful for real-time training to tightly synchronized, guaranteed event order mechanisms appropriate for analytical simulation.

Last, the RTI provides functions, which support coordination of a distributed simulation execution, including distributed start/stop, pause/resume and checkpoint/restart commands.

4.2.1.4 Evaluation

HLA has successfully linked hundred of simulations supporting thousands of entities and highly dynamic weather, terrain and command and control messaging. It is the leading technology in simulation application integration and so is a strong contender as the glue that holds together multiple simulations in the NAS domain.

In addition, the large DOD user base has fostered creation of a fairly large set of COTS and GOTS HLA products, including FOM development, data collection and exercise control tools. The continued funding of HLA development and of HLA-based programs within the DOD means that this set of tools will continue to grow in number and capability for the foreseeable future.

That having been said, large-scale application of HLA has to date been mostly in the real-time world. This is not due to a bias in HLA capabilities in that direction; rather, it reflects the operational and funding priorities of the DOD. What is does mean is that the RTI implementations have not for the most part been stress-tested in large fast-time applications and that the RTI's defined capabilities are relatively immature in some areas of control for analytical simulations. The time and data management capabilities of the RTI were developed by experts from the parallel discrete event simulation (PDES) world and have the capabilities required for analytical simulation. The RTI offers less capability where it comes to the sorts of distributed initialization and multiple run management features that would be required for distributed Monte Carlo simulation. Existing analysis federations have found ways to augment base RTI capabilities to satisfy their requirements.

In all, despite the several blemishes described above, HLA remains the strongest contender in the industry for simulation integration for all three Use Cases.

Applicability to Use Cases:

- Fast-time Conceptual Trades – Applicable
- Fast-time Detailed Evaluation – Applicable
- Real-time Virtual Simulations – Applicable

4.3 Analysis Architectures

4.3.1 Aviation System Analysis Capability (ASAC)

4.3.1.1 Architectural Category

The Aviation System Analysis Capability (ASAC) is an integrated suite of models, execution management tools and databases for Fast-time Conceptual Trades.

Architectural Elements: Domain Models, Initialization, Data Collection and Analysis, Simulation Execution Management.

4.3.1.2 Characteristics

Aviation System Analysis Capability (ASAC)	
History	Developed starting in 1995 by LMI
User Community	As of 1999, 264 registered users across 58 government, industry and academic organizations
Cost	Unknown
Community Web Sites	www.asac.lmi.org
Documentation	Extensive architecture and design documentation
Software	Web-based versions of Executive Assistant (EA) and Quick Response System (QRS) software
Sources of information:	[Ro1999], [Ro1999b], [Ro1997]

4.3.1.3 Summary

The Aviation System Analysis Capability (ASAC) is an integrated suite of models, execution management tools and databases that is designed to evaluate the impacts of technology, policies, and procedures on the air transportation system. The system contains a set of models representing different aspects of the NAS domain. The models run independently but have compatible inputs and outputs and so can be chained together for complex analyses. To support chaining together models ASAC contains a fixed set of analysis templates. These are stereotypes of ways models can be combined for particular analyses. Figure 4.3.1.1 shows an example of an analysis template.

ASAC, in addition to its models and data, contains a set of analyst support tools. The primary analyst interface is the Executive Assistant (EA). The EA allows users to select templates, populate the templates with models and then execute sets of runs. Figure 4.3.1.2 shows the components of the ASAC EA. The EA User Application provides the user interface to the analyst. The Model and Application Service components handle the details of selecting and executing models. The Analysis Application provides control over an EA execution. The Analysis Application also allows automation in variation of parameters for sensitivity analysis, optimization on a parameter value and backsolving against a parameter.

The Quick Response System (QRS) provides a simpler interface to the models within

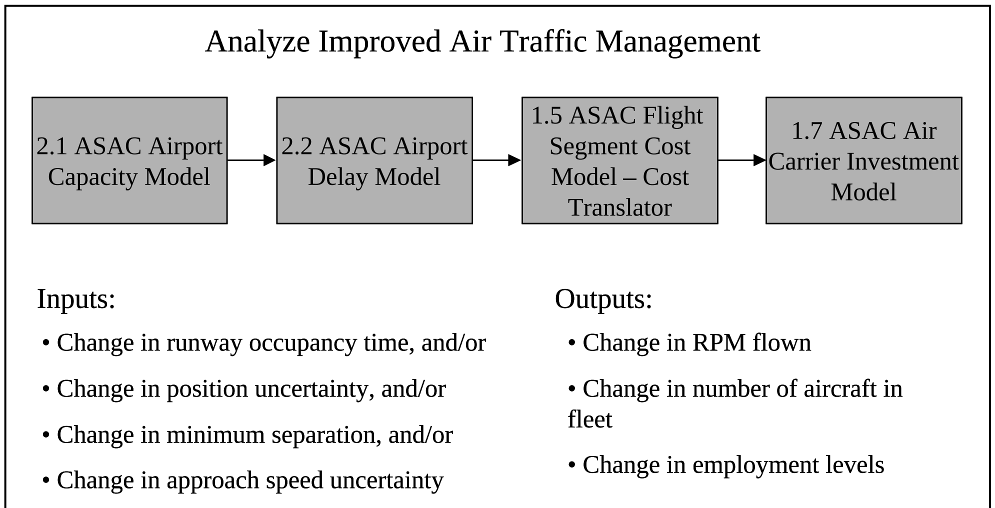

Analyze Improved Air Traffic Management
2.1 ASAC Airport Capacity Model
2.2 ASAC Airport Delay Model
1.5 ASAC Flight Segment Cost Model – Cost Translator
1.7 ASAC Air Carrier Investment Model
Inputs:
• Change in runway occupancy time, and/or
• Change in position uncertainty, and/or
• Change in minimum separation, and/or
• Change in approach speed uncertainty
Outputs:
• Change in RPM flown
• Change in number of aircraft in fleet
• Change in employment levels

Figure 4.3.1.1: Example ASAC Analysis Template

ASAC. QRS allows a user to parameterize and run an individual model and generate reports of model results. As its name suggests, the QRS is intended to provide a “quick and dirty” way of running models without going through the complexity of the EA.

ASAC also includes additional components for document management and for linking to external NASA sites and data.

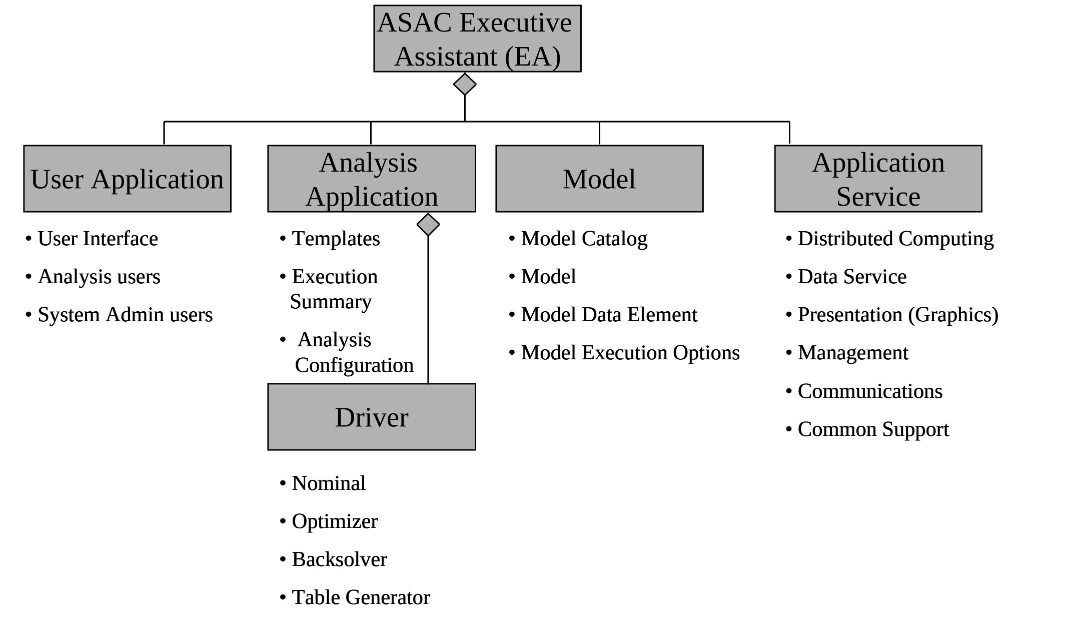


Figure 4.3.1.2: ASAC Executive Assistant Architecture

4.3.1.4 Evaluation

ASAC is an analysis architecture rather than a run-time architecture and so technically lies largely beyond the scope of this study. The system does not have a run-time architecture per se, in that it is really an execution shell that runs models one at a time in succession. The only run-time linkage is through common data definitions across the various models. In addition, while the system design is thoroughly documented via a domain-specific software architecture, it was not possible, given the lack of access to the ASAC software, to determine the capabilities of the present version of the software. It is not clear that the software is amenable to extension to incorporate interoperation with other systems.

All that having been said, the ASAC paradigm is useful to consider as a straightforward way of assembling and executing models for conceptual trades. The analysis architecture should take into consideration support for functions such as those provided by the Executive Assistant in the areas of multiple run management and rapid yet flexible composition of models for conceptual analysis.

Applicability to Use Cases:

- Fast-time Conceptual Trades – Applicable
- Fast-time Detailed Evaluation – Less Applicable

- Real-time Virtual Simulations – Not Applicable

4.3.2 AIRMM

4.3.2.1 Architectural Category

AIRMM is an architectural concept incorporating a full lifecycle of modeling analysis. The architecture is composed of a set of frameworks covering aspects of analytical tasks ranging from knowledge management to simulation execution and analysis.

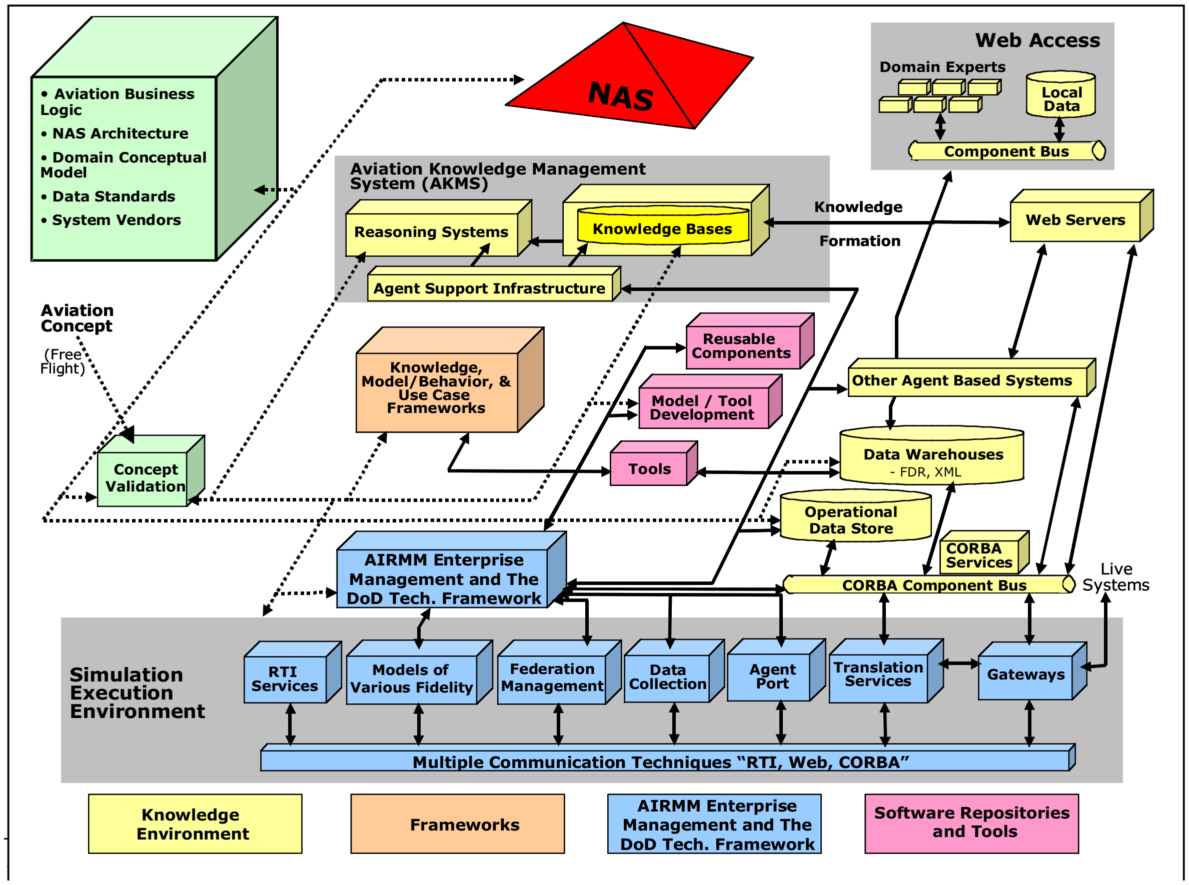
Architectural Elements: All reference architecture components.

4.3.2.2 Characteristics

AIRMM	
History	Developed at the FAA Tech Center based on requirements from the 1999 Aviation Modeling and Simulation Workshops
User Community	N/A
Cost	None
Community Web Sites	None. Project web site is http://www.tc.faa.gov/act-500/nasacb/airmm/airmm.html
Documentation	Several papers and web site
Software	Prototypes under development
Sources of information:	Web site, [SC2000]

4.3.2.3 Summary

In 1999 a series of workshops, the Aviation Modeling and Simulation Workshops (AMSW) were held to solicit input from the aviation community regarding requirements



for an improved aviation system level Air Traffic Management (ATM) modeling capability. The workshops identified a need for a flexible, integrated modeling capability to address issues and concepts from a top-down policy level to a detailed NAS architecture level. AIRMM was developed with the goal of satisfying this modeling and simulation need.

The AIRMM architecture is based on creating collections of systems, methodologies, standards, and tools. These components are envisioned as being organized into a set of frameworks which together compose AIRMM. The main AIRMM frameworks include:

- The Knowledge Formulation and Analysis Environment,
- Extendable frameworks for knowledge, model/behavior, and use cases,
- Run-time framework based on DOD's Technical Framework for Modeling and Simulation, and
- Repositories for software components and tools.

The complete set of AIRMM frameworks is shown in Figure 4.3.2.1. Of these, the run-time framework is most germane to the present study.

The AIRMM run-time framework is based on the DOD High Level Architecture,

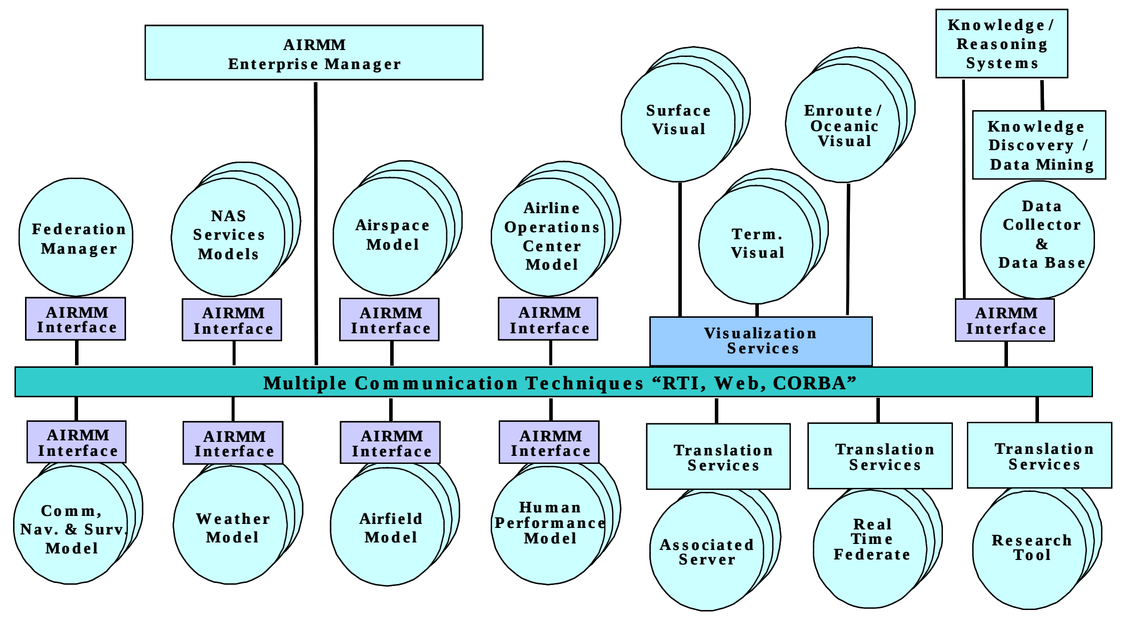


Figure 4.3.2.2: AIRMM Execution Environment

previously in Section 5.2. As shown in Figure 4.3.2.2, the AIRMM run-time is an HLA federation with communications extensions including CORBA and Web protocols. AIRMM creates a combined interface that unifies the HLA RTI, object persistence, an agent-based simulation platform and other elements.

The AIRMM federation includes individual federates for various NAS elements such as Air Operations Centers (AOC) as well as elements which surround the NAS, such as weather and surveillance. The AIRMM federation also includes typical HLA modules for run-time control, visualization and data collection, as well as “feedback optimization” functions to perform optimization over multiple runs a la the ASAC Executive Assistant. The architecture also calls out a Human Performance Model separate from the other models and simulations.

AIRMM run-time components incorporate several COTS/GOTS technologies. In addition to HLA, simulations in AIRMM are built upon a Java agent tool kit called JATLite and use the KQML knowledge representation language. Environmental elements are represented based on the DOD Synthetic Environment Data Representation Interchange Standard (SEDRIS) and command and control messaging uses the DOD Command and Control Simulation Interchange Language (CCSIL).

The other AIRMM frameworks deal with other elements of the simulation/analysis lifecycle. The Knowledge Formulation and Analysis Environment supports efficient formulation, capturing, and abstraction of aviation knowledge so that it is readily available to simulation developers and analysts. The Knowledge, Model/Behavior and Use Case frameworks have the goal of reducing the effort involved in structuring and management of aviation knowledge .The knowledge framework organized knowledge relative to the aviation domain and the analytical model base. The model/behavior framework provides a formalized mapping of aviation domain behaviors to an existing model base. and is used to capture behavior of the modeling components. The Use Case framework provides a standardized representation of modeling scenarios or use cases for modeling the aviation domain, with support for capturing standardized scenarios to verify and validate new modeling behavior.

Last, repositories provide a toolkit environment, allowing researchers to retrieve models, simulations and data based on descriptions expressed in terms of the Model/Behavior frameworks.

4.3.2.4 Evaluation

AIRMM is a far-reaching attempt to create a system for managing the tools, data and processes involved in the full simulation lifecycle. It mirrors and builds upon work done within the DOD, most notably the Joint Simulation System (JSIMS). It should be noted that JSIMS has expanded well beyond its original budget and schedule; implementation of the full AIRMM vision could be a similarly large undertaking.

The run-time component of AIRMM is, as previously mentioned, consistent with proven DOD technologies and is largely consistent with the architecture proposed in this study. There is in some of the AIRMM documentation a blurring between architecture and implementation, with, for example, a specific language (Java) specified for implementation of federation management software, specific allocation of domain elements (en route controllers, etc.) to particular simulations and universal adoption of somewhat heavyweight agent-oriented programming tools.

As a whole AIRMM is beyond the run-time orientation of this study and provides a large vision of NAS modeling and simulation. Within its run-time framework, it builds upon a proven foundation.

Applicability to Use Cases:

- o Fast-time Conceptual Trades – Applicable
- o Fast-time Detailed Evaluation – Applicable
- o Real-time Virtual Simulations – Applicable

4.4 Domain Modeling

The following are some domain modeling efforts that relate to the NAS domain.

4.4.1 FAA NIAC

The FAA's NAS Information Architecture Committee (NIAC) coordinates the establishment and future maintenance of information-based processes and procedures. NIAC has the goal of enabling interoperability of systems across the National Airspace System (NAS) via data standardization and data exchange. While the NIAC is geared towards actual systems rather than simulations, it can provide a foundation for development of domain models for the NAS simulation domain. [NI2001]

4.4.2 DOD Conceptual Model of the Mission Space

The Defense Modeling and Simulation Office's Conceptual Model of the Mission Space (CMMS) project represents the DOD's efforts in developing and documenting its domain models. Conceptual Models of the Mission Space are simulation implementation-independent functional descriptions of the real world processes, entities, and environment associated with a particular set of missions. In particular, CMMS is: 1.) A disciplined procedure by which the simulation developer is systematically informed about the real world problem to be synthesized. 2.) A set of information standards the simulation subject matter expert employs to communicate with and obtains feedback from the military operations subject matter expert. 3.) The real world, military operations basis for subsequent, simulation-specific analysis, design, and implementation, and eventually verification, validation, and accreditation/certification. 4.) A singular means for establishing re-use opportunities in the eventual simulation implementation by identifying commonality in the relevant real world activities. And 5.) A library of re-usable conceptual models for simulation development. [DM2001]

4.4.3 SISO Reference FOMs / BOMs

The Simulation Interoperability Standards Organization has coordinated efforts towards establishing reference FOMs, or data interchange standards, for communities within DOD. Reference FOM elements capture archetypes for interactions among simulation entities. These partial baseline FOMs are referred to as Base Object Models, or BOMs. A BOM consists of one or more classes, interactions, associated attributes, parameters, and relevant parent class data. BOMs provide a distinct way to represent individual simulation interaction patterns and components that can be used to build or modify a federation or interoperable environment. The Basic Premise is that the design and development of interoperable environment" should begin with the exploration of reusing available interoperable simulation object model patterns and components (a.k.a. BOMs). [Gu2001]

5 Conclusions

5.1 Overall Architecture

The general conclusion of this study is that the three Use Cases can be satisfied with one architecture, with appropriate tuning for particular demands of each Use Case.

The key element is that, as described in Section 2, all of the scenarios have the potential to be *distributed*, that is, run across multiple computers, and they share the need to be able to *compose* simulations to create an integrated, multi-faceted representation of the NAS. Thus, the simulation architecture should be built from a basis of interoperability and integration, including integration of legacy software

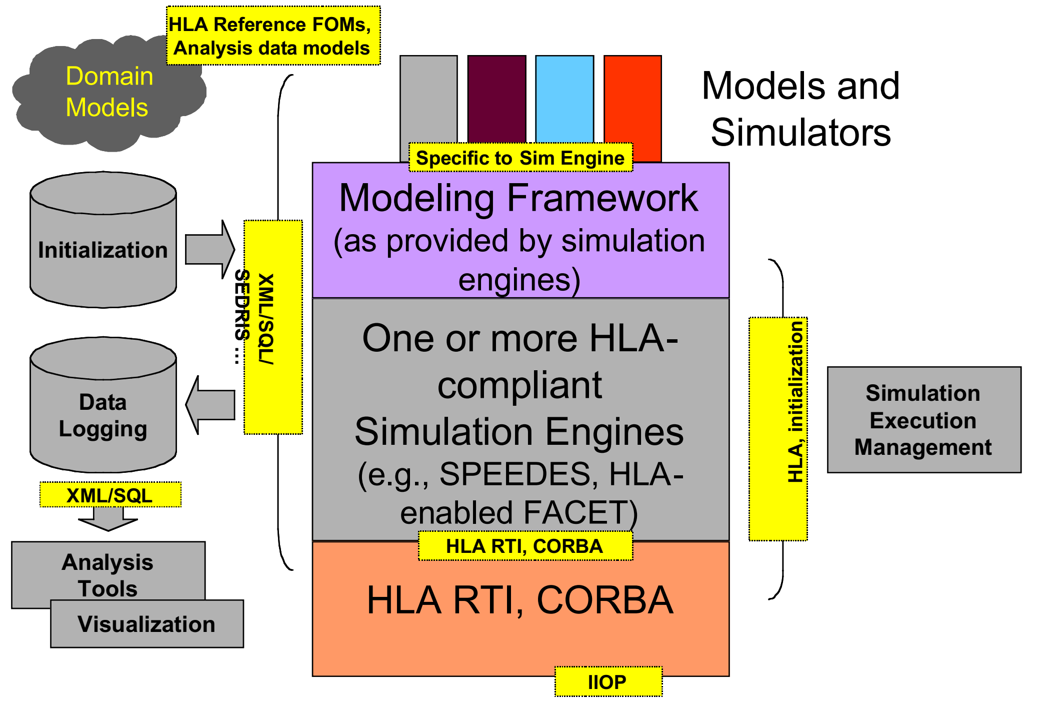

HLA Reference FOMs, Analysis data models
Domain Models
Models and Simulators
Specific to Sim Engine
Modeling Framework (as provided by simulation engines)
Initialization
XML/SQL/ SEDRIS ...
Data Logging
One or more HLA-compliant Simulation Engines (e.g., SPEEDES, HLA-enabled FACET)
HLA, initialization
Simulation Execution Management
XML/SQL
HLA RTI, CORBA
Analysis Tools
Visualization
HLA RTI, CORBA
IIOP

Figure 5.1.1: Reference run-time architecture with candidate technologies

HLA is the most mature mechanism for integrating heterogeneous, distributed simulations, and so it is recommended that HLA be chosen as the backbone for distributed simulation. A number of the simulation engines examined are already compliant to one degree or another with HLA and so offer a foundation for building new simulations. HLA was specifically designed for legacy integration of legacy ATM simulations into HLA compliance is readily achievable. HLA also provides a proven process for community-based development of simulation standards and systems via the FEDEP. Thus, HLA is selected as the basis for run-time simulation integration.

The HLA RTI is not, however, the entire answer for analytical simulation integration. HLA is primarily a publish/subscribe paradigm, with the exception of particular control. Thus HLA offers no mechanism for remote method invocation (RMI), that is, for a function call interface between simulations. Thus, as shown in Figure 5.1.1, the

Integration/Communications layer of the architecture includes not only the RTI but also CORBA (alternatively, some other equivalent mechanism such as the Microsoft SOAP protocol could be used). RMI mechanisms could be used for, example, to allow simulations to query a weather server (e.g., “What is the current visibility at SFO?”) rather than have the weather server publish all of its weather information (e.g., “Here is visibility information for all airports across the U.S.”). RMI can also be used in multi-run initialization and other functions.

The last integration/communications mechanism is SQL, via ODBC or equivalent. Databases by their very nature provide powerful data storage capabilities which can be essential to management of large analytical data sets. Strict interpretation of HLA Rules requires all FOM (that is, public) data to be passed via the RTI, meaning that only data collection applications would be integrated with a database, however, in practice this rule may be relaxed to allow other federates to load data directly from a database. The main caveat in this area is that guaranteed ordering of data, that is, making sure that events and data are time-tagged and stored in the order they transpired in the simulation can only be achieved by coordinated data capture through the RTI and so individual federates should not store their own data if maintenance of exact temporal ordering of collected data is required.

There are two small caveats regarding HLA. As mentioned in Section 4.2.1, the HLA has been applied more often in real-time than in fast-time applications. Thus, the available COTS/GOTS generally lack the multi-run management features equivalent to those of the ASAC Executive Assistant (see Section 4.3.1); these will have to be fleshed out for the fast-time world. HLA also has some minor tweaks in terms of repeatable execution. While all data, time and event traffic passed through the RTI can be made repeatable through guaranteed ordering, some of the functions of the RTI itself, such as ownership and declaration management, are not integrated with the time/event management mechanisms. Thus, simulations which dynamically create entities and transfer ownership of entities from one simulation to another may not be exactly repeatable using current RTI technologies.

The remaining reference architecture components can be at least partially satisfied with existing technologies, as previously summarized in Figure 5.1. It is recommended that, for the sake of reuse, a small number of modeling frameworks / simulation engines (e.g., SPEEDES for high performance computing plus one simpler framework for smaller simulation development) be adopted as the preferred tools for the community. Because of the neutral nature of HLA the community would not, however be limited to using only these engines.

Figure 6.1.1 once again shows the reference architecture, in this case with particular technologies applied to the components of the architecture.

5.2 Use Case 1: Fast-time Detailed Analysis

This Use Case requires a potentially large number of computers to handle the fidelity, scenario size and multi-faceted representations that would be typical of this type of analysis.

Figure 5.2.1 shows the reference architecture tailored to this Use Case. In this instance of the architecture the federation consists of a large number of federates synchronized through HLA. As shown in the figure, the federation may be composed of many different types of simulations and may be built on different simulation engines, some of which may themselves manage multiple processors (ref. Simulation Engine "B" in the figure). The federation may include both native (or adapted) HLA simulations and wrapped legacy simulations. The federation would use HLA's time management services to coordinate virtual time and would most likely be running as fast as possible without trying to maintain any relationship to wall-clock time. The HLA RTI would also be used for data distribution among simulations.

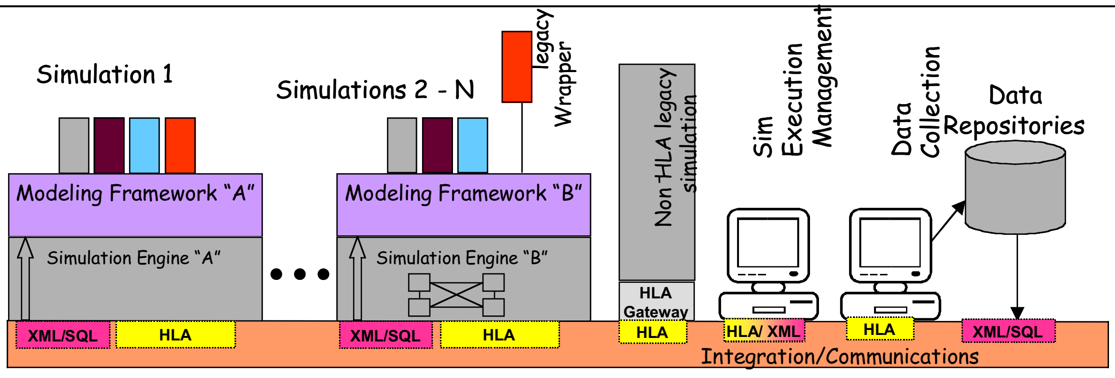


Figure 5.2.1: Architecture as implemented for Fast-time Detailed Analysis

Since the volume of data to be collected may be large the Data Collection federate would be its own computer or computers. In the STOW 97 system, the data collection federate consisted of four data collection computers, each of which subscribed to and locally stored a subset of the overall data stream. These computers were connected to a single large database server; periodically each data collection machine would bulk load its data into the database. This setup successfully supported a "collect everything" approach where any and all simulation data could be queried *ad hoc* at run-time.

Simulation Execution Management would use HLA where possible. SEM may in additional use RMI calls to the various federates as well as agents executing on each machine to control execution of the federates and distribute non-run-time data (such as initialization files), which would be encoded using XML. While it is not shown explicitly in the diagram, RMI could also be used from within federates to gain access to servers and utilities

5.3 Use Case 2: Fast-time Conceptual Trades

Fast-time Conceptual Trades share many characteristics with Fast-time Detailed Analyses and so the architectural instance shown in Figure 6.2.1 applies to this case as well. In this Use Case the number of simultaneously executing simulations will typically be smaller. The volume of data will be smaller as well and so in practice the multiple machines shown in the diagram may collapse down to a small number, with Simulation Execution Management and Data Collection sharing a host. For truly small simulations, the entire federation may be executed on a single computer.

In current systems such as ASAC, conceptual tradeoff models are run not in parallel but in series, integrated through common data. The proposed architecture supports this methodology as well. Figure 5.3.1 shows how this would be implemented. Each simulation in the chain would execute as its own small federation. These federations would execute under the control of a SEM program that would persist across the multiple federation executions. As in ASAC, chained models would be integrated by common data definitions (indicated by the dashed arrow in the figure). Unlike the ASAC model, each simulation program could be arbitrarily complex, where any one of the federation executions in the chain could contain multiple federates.

As with the first Use Case, each federation execution would use HLA's time management services to coordinate virtual time and data distribution.

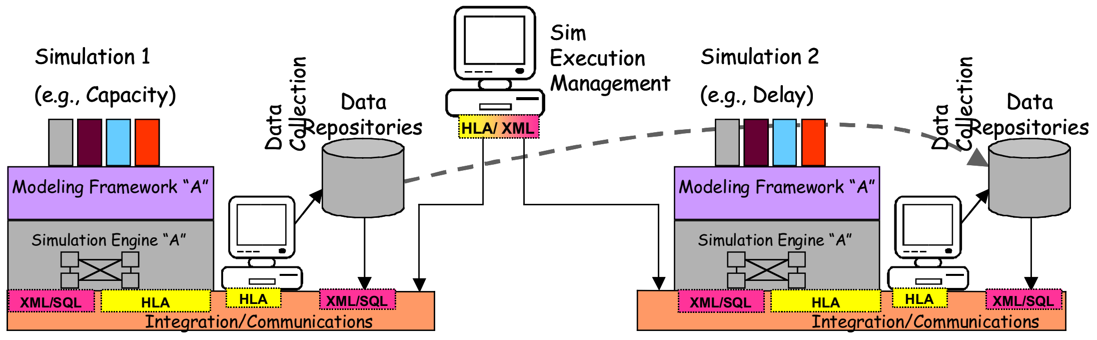


Figure 5.3.1: Architecture as implemented for sequential simulation execution of Fast-time Conceptual Trades

5.4 Use Case 3: Real-time Virtual Simulators

Real-time Virtual Simulations have somewhat different characteristics than the previous two Use Cases. A Virtual federation typically consists of a small number of very high fidelity human-in-the-loop simulators augmented by a small number of simulations. Figure 6.4.1 shows the architecture as implemented for this case.

In this instance the simulators are assumed to be in a sense "closed source" software, that is, not easily amenable to modification. Consequently, while they are shown integrating at run-time via HLA, they are not shown as having been adapted to accept XML-formatted initialization data. This is usually the case for virtual trainers; immersive simulators used for human factors analyses may have more open source code and so may be amenable to greater integration.

In a Real-time federation, federates may elect not to use the RTI's time management functions, instead running in loose synchronization through wall-clock time. Alternately, the may operate with one federate regulating time, where that federate controls the

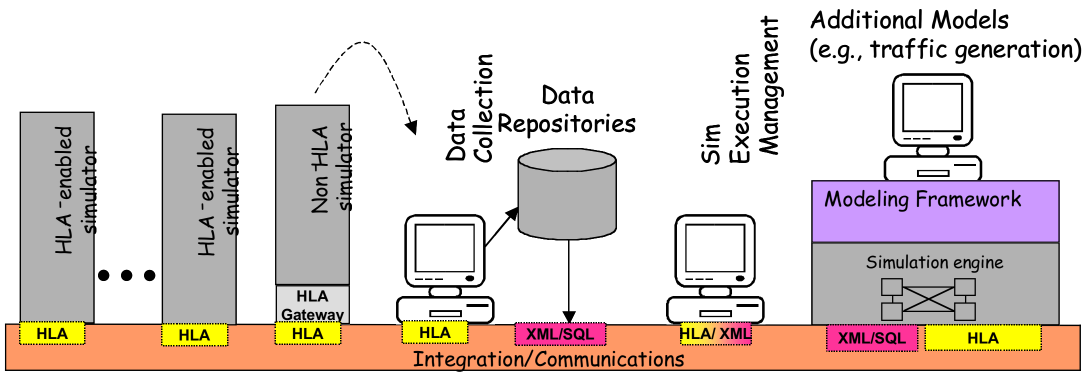


Figure 5.4.1: Architecture as implemented for Real-time Virtual Simulations

synchronization with wall-clock. In this case the other federates would be subordinate to the regulating federate. A Real-time federation may also elect, based on anticipated network bottlenecks, to use best-effort rather than guaranteed delivery for some or all messaging so as to avoid slowing down the federation due to communications delays.

Data collection in this environment typically involves less data than Use Case 1 and so the data collector, while executing on its own hardware, will most likely not be overly complex. The overall analysis may include collection of streams of data such as video or audio. The RTI has been used to pass digitized audio data, however multimedia streams use a great deal of bandwidth and so it may make sense to collect these streams separately, perhaps time-tagged based on federation time. Figure 6.4.1 alludes to collection of data outside of HLA this via the bent arrow leading to data collection; this arrow indicates and alternate path of data collection – not for FOM data, but for other types of analysis measures.

5.5 Recommended Next Steps

To move forward with implementation of the NAS-wide simulation architecture it is important to take four steps. These steps, namely, *implement prototypes*, *build community*, *define data*, and *address issues* may be addressed in series or in parallel.

First, this study has provided only a first cut at requirements for NAS domain use cases. Many other considerations such as overall computational requirements, constraints from within the user community, and so on must be addressed in greater detail as a system is developed. The best way, however, to gain experience with the architecture is through implementation. Successively more capable prototypes will help refine the architecture and identify challenges particular to the domain.

Second, an underlying assumption of this study is that one of the goals of the NAS simulation architecture is to build a community wherein many simulationists contribute to an overall capability. Within the DOD it took a number of years to convince people of the value of this approach, which carries with it the perceived downside of imposing constraints on each developer which only serve to benefit other projects. DOD addressed this through the creation of workshops where its simulation community collaboratively developed standards and consequently developed a community focus and consensus[5]. The NAS simulation community may be smaller and less far-flung than that of DOD, however an architecture-based approach will be most successful if it is a shared vision. One way to accomplish this is to begin a discussion within the community of the appropriate domain models, Federation Object Models, XML DTDs and so on based on analysis scenarios such as DAG-TM.

Last, as mentioned before the proposed architecture, while believed to be the best available, has some areas of immature capability. There has been relatively little investment within the DOD in distributed scenario setup/simulation initialization tools and Monte Carlo multiple-run control. The NAS simulation community needs to begin to identify and address those issues. This can be done in concert with DOD, which is beginning to face those same issues in its simulation-based acquisition programs and in concert with HLA tool vendors. If the NAS simulation community intends to build community-wide repositories of models and data then it needs to evaluate model management technologies (see, for example [Ar2000]). Finally, as already stated, run-time is only one piece of the simulation lifecycle and so development of the simulation run-time architecture should be harmonized with related efforts in the pre- and post- run-time areas.

[5] When HLA was developed, DOD also encouraged its acceptance by mandating its use, however it was born of an extant consensus of what the next step was in simulation.

Glossary and Acronyms

AATT	Advanced Air Transportation Technologies
ATM	Air Traffic Management
AvSTAR	Aviation System Technology Advanced Research
BOM	Base Object Model (see also Federation Object Model)
Cluster Computing	A mechanism for creating high performance virtual computers (up to super-computer level) by interconnecting clusters of smaller computers, often through distributed shared memory.
CMMS	Conceptual Model of the Mission Space
Composition, Composable	The ability to create a larger whole (in this case, a broader or richer simulation representation) by combining various pieces of software into a jointly executable whole.
CORBA	Common Object Request Broker Architecture. A standard for remote method invocation. Also offers various defined service interfaces.
COTS	Commercial Off the Shelf
CPU	Central Processing Unit
CPU	Central Processing Unit
DAG-TM	
Distributed Shared Memory	A mechanism for extending virtual memory to multiple machines, allowing the physical memory of multiple computers to be accessed as a single logical block of memory.
DOF	Degrees of Freedom
DTD	Data Type Descriptor within XML
ETMS	Enhanced Traffic Management System. An FAA database of flight data information.
FEDEP	HLA Federation Development and Execution Process
Federate	A simulation that is a member of an HLA Federation.
Federation	A collection of simulations which interoperate via the HLA RTI, using an agreed upon set of data definitions (FOM).
Federation Object Model (FOM)	A data model used to define the data which can be exchanged by a set of HLA simulations. Provides a standard for data interoperability.
Gateway	A program or computer which serves to interconnect hardware or software with differing protocols, for example, to connect a virtual simulator with HLA-based simulations.
GOTS	Government Off the Shelf
High Level Architecture	See HLA
HLA	Department of Defense High Level Architecture for Modeling and Simulation. A collection of processes, interface specifications and software for simulation interoperability.
Multi-processor kernel	A simulation engine containing features such as optimistic scheduling which explicitly attempt to enhance performance by making use of multiple processors which may be available on a computer.
NAS	National Airspace System
Optimistic Scheduling	A mechanism for coordinating a computer simulation program running across a multi-processor machine which allows one or more processors to run ahead in time, knowing they may need to redo their computations based on the results of computations on other processors.
QAT	Quiet Aircraft Technology
Remote Method Invocation	The ability for a computer program to invoke software running in a different computer process or on a different computer as if it were a local function call.

RMI	Remote Method Invocation. A mechanism for a piece of software to invoke a function executing in a different software process.
RTI	Run-time Infrastructure. The software component of HLA (see HLA), offering time management, event and data management and other inter-simulation coordination functions.
RUC	Rapid Update Cycle. A weather data format.
SEDRIS	Synthetic Environment Data Representation and Interchange Specification
SEM	Simulation Execution Management
Single-processor kernel	A simulation engine which does not contain any features which explicitly attempt to enhance performance by making use of multiple processors which may be available on a computer.
SISO	Simulation Interoperability Standards Organization
SOAP	Simple Object Access Protocol. A method for remote method invocation based on web protocols (HTTP).
SOM	HLA Simulation Object Model
Symmetric Multi-processor (SMP)	A computing architecture characterized by tightly-coupled series of identical processors, usually operating on a single shared bank of memory. The parallel nature of the machine is typically hidden from the user, with the operating system managing allocation of processor time to programs.
Time Warp	An optimistic scheduling algorithm for multi-processor computing.
Virtual Memory	Virtual memory is a technique that allows processes that may not be entirely in the memory to execute by means of automatic storage allocation upon request. The term virtual memory refers to the abstraction of separating logical memory--memory as seen by the process—from physical memory--memory as seen by the processor
XML	Extensible Markup Language. A standard mark-up language for data description. Used to exchange data between applications.

Bibliography

[Ar2000] Aronson, J. and D. Wade. Benefits and Pitfalls in Composable Simulation. *Proc. Spring 2000 Simulation Interoperability Workshop.* Paper 00S-SIW-155. March 2000. Orlando, FL.

[Be1997] Bennett, P.A. Rapid Development and Rehosting of Dynamic Graphical Interfaces. October 1997. Available at http://www.virtualprototypes.com/products_solutions/PDF/rehostingbennet.pdf.

[Da1999] Daniels, M. Integrating Simulation Technologies with Swarm. Available at http://www.santafe.edu/~mgd/anlchicago.html.

[De2001] Distributed Environment for Simulation, Rapid Engineering and Experimentation overview paper. Available at http://155.178.136.28/rdhfl/desireetxt.htm.

[DM1998] Defense Modeling and Simulation Office. HLA Interface Specification Version 1.3. 20 April 1998. Available at http://www.dmso.mil/briefs/war/hla/specs/main_body.pdf.

[DM1999] Defense Modeling and Simulation Office. HLA Federation Development and Execution Process (FEDEP) Model. Version 1.5. 8 December 1999. Available at http://www.dmso.mil/briefs/msdocs/guide/fedepv15.pdf.

[DM2001] Defense Modeling and simulation Office. Conceptual Model of theMission Space Home Page. See http://www.dmso.mil/index.php?page=75.

[Er2001] Erzberger, H. The Automated Airspace Concept. Paper #160, 4th USA/Europe Air Traffic Management R&D Seminar. Dec 3-7, 2001. Santa Fe, NM. Available at http://atm2001.eurocontrol.fr.

[Fu1993] Fujimoto, R. M. Parallel discrete event simulation: Will the field survive? *ORSA Journal on Computing.* 5(3):213-230, Summer 1993.

[Gu2001] Gustavson, P., et. al. BOM Study Group Final Report. SISO-REF-005-2001. 15 May 2001. Available at http://www.sisostds.org.

[Hi2001] Hillman, R., Hanna, J. and Walter, M. Modeling and Simulation of the Joint Battlespace Infosphere Scalability. *MSIAC M&S Journal Online.* Vol. 3, No. 1. Available at http://www.msiac.dmso.mil.

[Ip2000] Ippolito, C.A. and Pritchett, A.R. Software Architecture for a Reconfigurable Flight Simulator. *Proc. AIAA Modeling and Simulation Technologies Conference*. 14-17 August, 2000. Denver, CO.

[Ma2001] VR-Link Product Brochure. Available at http://www.mak.com/vrlink.pdf.

[NI2001] NIAC Home Page. See http://www.faa.gov/niac/.

[OS1998] Osim Simulation Module; A Commercial Simulation Development Product. March 1998. Available at http://www.originalsim.com.

[OS1998b] The Osim S/M Architecture/ January 1998. Available at http://www.originalsim.com.

[OS2001] OriginalSim, Inc. Re-Usability of Legacy Software in an Object-Oriented Application Framework. White Paper. Available at http://www.originalsim.com

[Pa2001] Pace, Dale. Simulation Conceptual Model Role in Determining Compatibility of Candidate Simulations for a HLA Federation. *Proc. Spring 2001 Simulation Interoperability Workshop*. Paper 01S-SIW-024. 25-30 March 2001. Orlando, FL. Available at http://www.dmso.mil/briefs/war/vva/prod/01S-SIW-024.doc.

[Pr2000] Pritchett, A. R., Lee, S., et. al. Hybrid-System Simulation for National Airspace System

Safety Analysis. *Proc. 2000 Winter Simulation Conference*. 10-13 December 2000.

[Pr2001] Pritchett, A., et. al. Examining Air Transportation Safety Issues Through Agent-Based Simulation Incorporating Human Performance Models. Submitted for publication to AIAA Journal of Aircraft.

[Ro1997] Roberts, E. and Villani, J. ASAC Executive Assistant Architecture Description Summary. NASA Contractor Report 201681. April 1997.

[Ro1999] Roberts, E. and Kostiuk, P. Aviation System Analysis Capability. Executive Assistant Analyses. NASA/CR-1999-209118. March 1999.

[Ro1999b] Roberts, E., et. al. Aviation System Analysis Capability. Executive Assistant Development. NASA/CR-1999-209119. March 1999.

[Sc2000] Schwartz, A. and Richards, J. The Aviation Integrated Reasoning Modeling Matrix (AIRMM). December 2000.

[SD2000] Swarm Development Group. A Tutorial Introduction to Swarm. Available at http://www.swarm.org/csss-tutorial.

[SE2001] See http://www.sedris.org.

[SP2000] SPEEDES User's Guide. 17 November 2000. Document S024. Revision 2. Available at http://www.speedes.com.

[St1999] Steinman, J. S., et. al. Design of the HPC-RTI for the High-Level Architecture. *Proc. Fall 1999 Simulation Interoperability Workshop*. Paper 99F-SIW-067. 12-17 September 1999. Orlando, FL.

[TR2000] TRANSIMS Press Release. Available at http://www.lanl.gov/orgs/pa/News/100500.html#anchor1.

[TR2001] TRANSIMS Web site, http://transims.tsasa.lanl.gov.

[TR2001b] TRANSIMS: Transportation Analysis Simulation System. Version 2.0 Documentation. Volume 3, Modules. LANL Document LA-UR-00-1725. Available at [TR2001]